Introduction to
Sound Processing

Davide Rocchesso*

* Università di Verona
Dipartimento di Informatica
email: D.Rocchesso@computer.org
www: http://www.scienze.univr.it/~rocchess

The book is accessible from the author's web site: http://www.scienze.univr.it/~rocchess.
The book is listed in http://www.theassayer.org, where reviews can be posted.

ISBN 88-901126-1-1

Cover Design: Claudia Calvaresi.
Editorial Production Staff: Nicola Bernardini, Federico Fontana, Alessandra Ceccherelli, Nicola Giosmin, Anna Meo.

Produced from LaTeX text sources and PostScript and TIFF images.
Compiled with VTEX/free.
Online distributed in Portable Document Format.
Printed and bound in Italy by PHASAR Srl, Firenze.

Contents

Preface

What you have in your hands, or on your screen, is an introductory book on sound processing. By reading this book, you may expect to acquire some knowledge on the mathematical, algorithmic, and computational tools that I consider to be important in order to become proficient sound designers or manipulators.

The book is targeted at both science- and art-oriented readers, even though the latter may find it hard if they are not familiar with calculus. For this purpose an appendix of mathematical fundamentals has been prepared in such a way that the book becomes self contained. Of course, the mathematical appendix is not intended to be a substitute of a thorough mathematical preparation, but rather as a shortcut for those readers that are more eager to understand the applications.

Indeed, this book was conceived in 1997, when I was called to teach introductory audio signal processing in the course "Specialisti in Informatica Musicale" organized by the Centro Tempo Reale in Firenze. In that class, the majority of the students were excellent (no kidding, really superb!) music composers. Only two students had a scientific background (indeed, a really strong scientific background!). The task of introducing this audience to filters and trasforms was so challenging for me that I started planning the lectures and laboratory material much earlier and in a structured form. This was the initial form of this book. The course turned out to be an exciting experience for me and, based on the music and the research material that I heard from them afterward, I have the impression that the students also made good use of it.

After the course in Firenze, I expanded and improved the book during four editions of my course on sound processing for computer science students at the University of Verona. The mathematical background of these students is different from that of typical electrical engineering students, as it is stronger in discrete mathematics and algebra, and with not much familiarity with advanced

and applied calculus. Therefore, the books presents the basics of signals, systems, and transforms in a way that can be immediately used in applications and experienced in computer laboratory sessions.

This is a free book, thus meaning that it was written using free software tools, and it is freely downloadable, modifiable, and distributable in electronic or printed form, provided that the enclosed license and link to its original web location are included in any derivative distribution. The book web site also contains the source codes listed in the book, and other auxiliary software modules.

I encourage additions that may be useful to the reader. For instance, it would be nice to have each chapter ended by a section that collects annotations, solutions to the problems that I proposed in footnotes, and other problems or exercises. Feel free to exploit the open nature of this book to propose your additional contents.

Venezia, 11th February 2004 Davide Rocchesso

Chapter 1

Systems, Sampling and Quantization

1.1 Continuous-Time Systems

Sound is usually considered as a mono-dimensional signal (i.e., a function of time) representing the air pressure in the ear canal. For the purpose of this book, a Single-Input Single-Output (SISO) System is defined as any algorithm or device that takes a signal in input and produces a signal in output. Most of our discussion will regard linear systems, that can be defined as those systems for which the superposition principle holds:

Superposition Principle : if y_1 and y_2 are the responses to the input sequences x_1 and x_2, respectively, then the input $ax_1 + bx_2$ produces the response $ay_1 + by_2$.

The superposition principle allows us to study the behavior of a linear system starting from test signals such as impulses or sinusoids, and obtaining the responses to complicated signals by weighted sums of the basic responses.

A linear system is said to be linear time-invariant (LTI), if a time shift in the input results in the same time shift in the output or, in other words, if it does not change its behavior in time.

Any continuous-time LTI system can be described by a differential equation. The Laplace transform, defined in appendix A.8.1 is a mathematical tool that is used to analyze continuous-time LTI systems, since it allows to transform complicated differential equations into ratios of polynomials of a complex

variable s. Such ratio of polynomials is called the transfer function of the LTI system.

Example 1. Consider the LTI system having as input and output the functions of time (i.e., the signals) $x(t)$ and $y(t)$, respectively, and described by the differential equation

$$\frac{dy}{dt} - s_0 y = x \ . \tag{1}$$

This equation, transformed into the Laplace domain according to the rules of appendix A.8.1, becomes

$$sY_L(s) - s_0 Y_L(s) = X_L(s) \ . \tag{2}$$

Here, as in most of the book, we implicitly assume that the initial conditions are zero, otherwise eq. (2) should also contain a term in $y(0)$. From the algebraic equation (2) the transfer function is derived as the ratio between the output and input transforms:

$$H(s) = \frac{1}{s - s_0} \ . \tag{3}$$

###

The coefficient s_0, root of the denominator polynomial of (3), is called the pole of the transfer function (or pole of the system). Any root of the numerator would be called a zero of the system.

The inverse Laplace transform of the transfer function is an equivalent description of the system. In the case of example 1.1, it takes the form

$$h(t) = \begin{cases} e^{s_0 t} & t \geq 0 \\ 0 & t < 0 \end{cases} \ , \tag{4}$$

and such function is called a causal exponential.

In general, the function $h(t)$, inverse transform of the transfer function, is called the impulse response of the system, since it is the output obtained from the system as a response to an ideal impulse[1].

The two equivalent descriptions of a linear system in the time domain (impulse response) and in the Laplace domain (transfer function) correspond to two alternative ways of expressing the operations that the system performs in order to obtain the output signal from the input signal.

[1] A rigorous definition of the ideal impulse, or Dirac function, is beyond the scope of this book. The reader can think of an ideal impulse as a signal having all its energy lumped at the time instant 0.

The description in the Laplace domain leads to simple multiplication between the Laplace transform of the input and the system transfer function:

$$Y(s) = H(s)X(s) \ . \tag{5}$$

This operation can be interpreted as multiplication in the frequency domain if the complex variable s is replaced by $j\Omega$, being Ω the real variable of the Fourier domain. In other words, the frequency interpretation of (5) is obtained by restricting the variable s from the complex plane to the imaginary axis. The transfer function, whose domain has been restricted to $j\Omega$ is called frequency response. The frequency interpretation is particularly intuitive if we imagine the input signal as a complex sinusoid $e^{j\Omega_0 t}$, which has all its energy focused on the frequency Ω_0 (in other words, we have a single spectral line at Ω_0). The complex value of the frequency response (magnitude and phase) at the point $j\Omega_0$ corresponds to a joint magnitude scaling and phase shift of the sinusoid at that frequency.

The description in the time domain leads to the operation of convolution, which is defined as[2]

$$y(t) \stackrel{\triangle}{=} (h * x)(t) = \int_{-\infty}^{+\infty} h(t-\tau)x(\tau)d\tau \ . \tag{6}$$

In order to obtain the signal coming out from a linear system it is sufficient to apply the convolution operator between the input signal and the impulse response.

1.2 The Sampling Theorem

In order to perform any form of processing by digital computers, the signals must be reduced to discrete samples of a discrete-time domain. The operation that transforms a signal from the continuous time to the discrete time is called sampling, and it is performed by picking up the values of the continuous-time signal at time instants that are multiple of a quantity T, called the sampling interval. The quantity $F_s = 1/T$ is called the sampling rate.

The presentation of a detailed theory of sampling would take too much space and it would become easily boring for the readership of this book. For a more extensive treatment there are many excellent books readily available,

[2]The convolution will be fully justified for discrete-time systems in section 1.4. Here, for continuous-time systems, we give only the definition.

from the more rigorous [66, 65] to the more practical [67]. Luckily, the kernel of the theory can be summarized in a few rules that can be easily understood in terms of the frequency-domain interpretation of signals and systems.

The first rule is related to the frequency representation of discrete-time variables by means of the Fourier transform, defined in appendix A.8.3 as a specialization of the Z transform:

Rule 1.1 *The Fourier transform of a function of discrete variable is a function of the continuous variable ω, periodic*[3] *with period 2π.*

The second rule allows to treat the sampled signals as functions of discrete variable:

Rule 1.2 *Sampling a continuous-time signal $x(t)$ with sampling interval T produces a function $\hat{x}(n) \stackrel{\triangle}{=} x(nT)$ of the discrete variable n.*

If we call spectrum of a signal its Fourier-transformed counterpart, the fundamental rule of sampling is the following:

Rule 1.3 *Sampling a continuous-time signal with sampling rate F_s produces a discrete-time signal whose frequency spectrum is a periodic replication of the spectrum of the original signal, and the replication period is F_s. The Fourier variable ω for functions of discrete variable is converted into the frequency variable f (in Hz) by means of*

$$\omega = 2\pi f T = \frac{2\pi f}{F_s} \ . \tag{7}$$

Fig. 1 shows an example of frequency spectrum of a signal sampled with sampling rate F_s. In the example, the continuous-time signal had all and only the frequency components between $-F_b$ and F_b. The replicas of the original spectrum are sometimes called images.

Given the simple rules that we have just introduced, it is easy to understand the following Sampling Theorem, introduced by Nyquist in the twenties and popularized by Shannon in the forties:

Theorem 1.1 *A continuous-time signal $x(t)$, whose spectral content is limited to frequencies smaller than F_b (i.e., it is band-limited to F_b) can be recovered from its sampled version $\hat{x}(n) = x(nT)$ if the sampling rate $F_s = 1/T$ is such that*

$$F_s > 2F_b \ . \tag{8}$$

[3] This periodicity is due to the periodicity of the complex exponential of the Fourier transform.

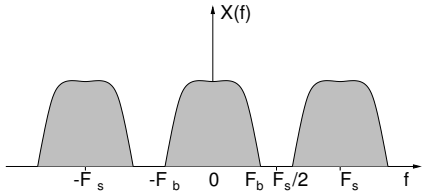

Figure 1: Frequency spectrum of a sampled signal

It is also clear how such recovering might be obtained. Namely, by a linear reconstruction filter capable to eliminate the periodic images of the base band introduced by the sampling operation. Ideally, such filter doesn't apply any modification to the frequency components lower than the Nyquist frequency, defined as $F_N = F_s/2$, and eliminates the remaining frequency components completely.

The reconstruction filter can be defined in the continuous-time domain by its impulse response, which is given by the function

$$h(t) \triangleq sinc(t) = \frac{\sin(\pi t/T)}{\pi t/T} , \tag{9}$$

which is depicted in fig. 2.

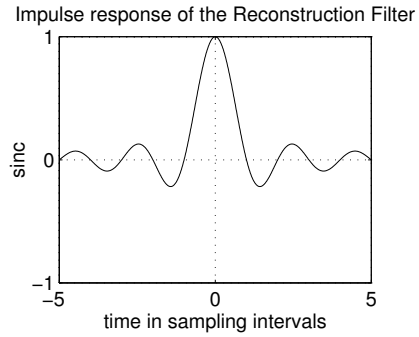

Figure 2: $sinc$ function, impulse response of the ideal reconstruction filter

Ideally, the reconstruction of the continuous-time signal from the sampled signal should be performed in two steps:

- Conversion from discrete to continuous time by holding the signal constant in time intervals between two adjacent sampling instants. This is achieved by a device called a holder. The cascade of a sampler and a holder constitutes a sample and hold device.

- Convolution with an ideal $sinc$ function.

The $sinc$ function is ideal because its temporal extension is infinite on both sides, thus implying that the reconstruction process can not be implemented exactly. However, it is possible to give a practical realization of the reconstruction filter by an impulse response that approximates the $sinc$ function.

Whenever the condition (8) is violated, the periodic replicas of the spectrum have components that overlap with the base band. This phenomenon is called aliasing or foldover and is avoided by forcing the continuous-time original signal to be bandlimited to the Nyquist frequency. In other words, a filter in the continuous-time domain cuts off the frequency components exceeding the Nyquist frequency. If aliasing is allowed, the reconstruction filter can not give a perfect copy of the original signal.

Usually, the word aliasing has a negative connotation because the aliasing phenomenon can make audible some spectral components which are normally out of the frequency range of hearing. However, some sound synthesis techniques, such as frequency modulation, exploit aliasing to produce additional spectral lines by folding onto the base band spectral components that are outside the Nyquist bandwidth. In this case where the connotation is positive, the term foldover is preferred.

1.3 Discrete-Time Spectral Representations

We have seen how the sampling operation essentially changes the nature of the signal domain, which switches from a continuous to a discrete set of points. We have also seen how this operation is transposed in the frequency domain as a periodic replication. It is now time to clarify the meaning of the variables which are commonly associated to the word "frequency" for signals defined in both the continuous and the discrete-time domain. The various symbols are collected in table 1.1, where the limits imposed by the Nyquist frequency are also indicated. With the term "digital frequencies" we indicate the frequencies of discrete-time signals.

Appendix A.8.3 shows how it is possible to define a Fourier transform for functions of a discrete variable. Here we can re-express such definition, as

Nyquist Domain					Symbol	Unit	
$[-F_s/2$	$\ldots$	0	$\ldots$	$F_s/2]$	f	[Hz] = [cycles/s]	
$[-1/2$	$\ldots$	0	$\ldots$	$1/2]$	f/F_s	[cycles/sample]	digital
$[-\pi$	$\ldots$	0	$\ldots$	$\pi]$	$\omega = 2\pi f/F_s$	[radians/sample]	freqs.
$[-\pi F_s$	$\ldots$	0	$\ldots$	$\pi F_s]$	$\Omega = 2\pi f$	[radians/s]	

Table 1.1: Frequency variables

a function of frequency, for discrete-variable functions obtained by sampling continuous-time signals with sampling interval T. This transform is called the Discrete-Time Fourier Transform (DTFT) and is expressed by

$$Y(f) = \sum_{n=-\infty}^{+\infty} y(nT) e^{-j2\pi \frac{f}{F_s} n} \ . \tag{10}$$

We have already seen that the function $Y(f)$ is periodic[4] with period F_s. Therefore, it is easy to realize that the DTFT can be inverted by an integral calculated on a single period:

$$y(nT) = \frac{1}{F_s} \int_{-F_s/2}^{F_s/2} Y(f) e^{j2\pi f nT} df \ . \tag{11}$$

In practice, in order to compute the Fourier transform with numeric means we must consider a finite number of points in (10). In other words, we have to consider a window of N samples and compute the discrete Fourier transform on that signal portion:

$$\hat{Y}(f) = \sum_{n=0}^{N-1} \hat{y}(n) e^{-j2\pi \frac{f}{F_s} n} \ . \tag{12}$$

In (12) we have taken a window of N samples (i.e., NT seconds) of the signal, starting from instant 0, thus forming an N-point vector. The result is still a function of continuos variable: the larger the window, the closer is the function to $Y(f)$. Therefore, the "windowing" operation introduces some loss of precision in frequency analysis. On the other hand, it allows to localize the analysis in the time domain. There is a tradeoff between the time domain and

[4]Indeed, the expression (10) can be read as the Fourier series expansion of the periodic signal $Y(f)$ with coefficients $y(nT)$ and components which are "sinusoidal" in frequency and are multiples of the fundamental $1/F_s$.

the frequency domain, governed by the Uncertainty Principle which states that the product of the window length by the frequency resolution Δf is constant:

$$\Delta f N = 1 \, . \tag{13}$$

Example 2. This example should clarify the spectral effects induced by sampling and windowing. Consider the causal complex exponential function in continuous time

$$y(t) = \begin{cases} e^{s_0 t} & t \geq 0 \\ 0 & t < 0 \end{cases} \, , \tag{14}$$

where s_0 is the complex number $s_0 = a + jb$. To visualize such complex signal we can consider its real part

$$\Re(y(t)) = \Re(e^{at} e^{jbt}) = e^{at} \cos(bt) \, , \tag{15}$$

and obtain fig. 3.a from it.

The Laplace transform of function (14) has been calculated in appendix A.8.1. It can be reduced to the Fourier transform by the substitution $s = j\Omega$:

$$Y(\Omega) = \frac{1}{j\Omega - s_0} \, . \tag{16}$$

The magnitude of the complex function (16) is drawn in solid line in fig. 3.

The sampled signal is also Fourier-transformable in closed form, by reducing the Z transform obtained in appendix A.8.3 by the substitution $z = e^{j\omega}$. The formula turns out to be[5]

$$Y(\omega) = \frac{1}{1 - e^{s_0/F_s} e^{-j\omega}} \, , \tag{17}$$

and its magnitude is drawn in dashed line in fig. 3 for $F_s = 50Hz$. We can see that sampling induces a periodic replication in the spectrum and that the periodicity is established by the sampling rate. The fact that the spectrum is not identically zero for frequencies higher than the Nyquist limit determines aliasing. This can be seen, for instance, in the heightening of the peak at the frequency of the damped sinusoid.

If we consider only the sampled signal lying within a window of $N = 7$ samples, we can compute the DTFT by means of (12) and obtain the third curve of fig. 3. Two important artifacts emerge after windowing:

[5] If we compare this formula with (57) of the appendix A, we see that here the variable s_0 in the exponent is divided by F_s. Indeed, the discrete-variable functions of appendix A.8.3 correspond to signals sampled with unit sampling rate.

- The peak is enlarged. In general, we wave a main lobe for each relevant spectral component, and the width of the lobe might prevent from resolving two components that are close to each other. This is a loss of frequency resolution due to the uncertainty principle.
- There are side lobes (frequency leakage) due to the discontinuity at the edges of the rectangular window. Smaller side lobes can be obtained by using windows that are smoother at the edges.

Unfortunately, for signals that are not known analytically, the analysis can only be done on finite segments of sampled signal, and the artifacts due to windowing are not eliminable. However, as we will show in sec. 4.1.3, the tradeoff between width of the main lobe and height of the side lobes can be explored by choosing windows different from the rectangular one.

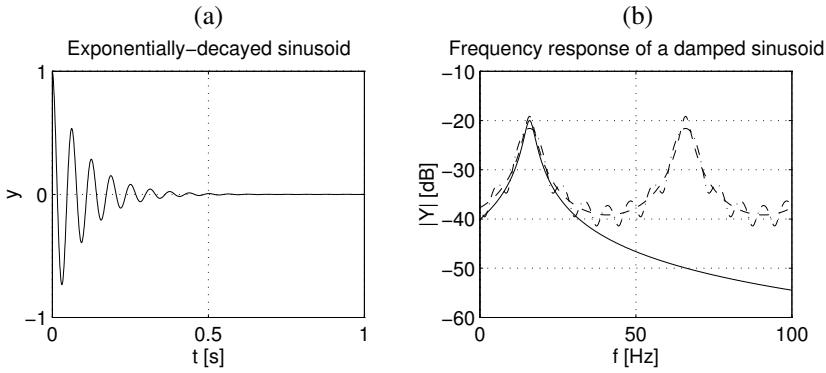

Figure 3: (a): Exponentially-decayed sinusoid, obtained as the real part of the complex exponential $y(t) = e^{s_0 t}$, with $s_0 = -10 + j100$; (b): Frequency analysis of the complex exponential $y(t) = e^{s_0 t}$. Transform of the continuous-time signal (continuous line), transform of the signal sampled at $F_s = 50Hz$ (dashed line), and transform of the sampled signal windowed with a 7-sample rectangular window (dash-dotted line)

To conclude the example we report the Octave/Matlab code (see the appendix B) that allows to plot the curves of fig. 3. The computation of the DTFT is particularly instructive. We have expressed the sum in (12) as a vector-matrix multiply, thus obtaining a compact expression that is computed efficiently. We also notice how Matlab and Octave manage vectors of complex numbers with the proper arithmetics.

```
% script that visualizes the effects of
% sampling and windowing
global_decl;
platform('octave'); %put either 'octave' or 'matlab'
a = - 10.0; b = 100;
s0 = a + i * b;
t = [0:0.001:1];
y = exp(s0*t); % complex exponential
subplot(2,2,1);  plot(t,real(y));
eval(mygridon);
title('Exponentially-decayed sinusoid');
xlabel('t [s]'); ylabel('y');
eval(myreplot);
pause;
f = [0:0.1:100];
Y = 1 ./ (i * 2 * pi * f - s0*ones(size(f)));
   % closed-form  Fourier transform
subplot(2,2,2); plot(f, 20*log10(abs(Y)), '-');
title('Frequency response of a damped sinusoid');
xlabel('f [Hz]'); ylabel('|Y| [dB]');
hold on;
Fs = 50;
Ysamp = 1 ./ (1 - exp(s0/Fs) * exp(- i*2*pi*f/Fs)) / Fs;
   % closed-form Fourier transform of the sampled signal
plot(f,20*log10(abs(Ysamp)),'--');
n = [0:6];
y = exp(s0*n/Fs);
Ysampw = y * exp(-i*2*pi/Fs*n'*f) / Fs;
   % Fourier transform of the windowed signal
   % obtained by vector-matrix multiply
plot(f,20*log10(abs(Ysampw)),'-.');  hold off;
eval(myreplot);
```

###

Finally, we define the Discrete Fourier Transform (DFT) as the collection of N samples of the DTFT of a discrete-time signal windowed by a length-N rectangular window. The frequency sampling points (called bins) are equally

spaced between 0 and F_s according to the formula

$$f_k = \frac{kF_s}{N} \ . \tag{18}$$

Therefore, the DFT is given by

$$Y(k) = \sum_{n=0}^{N-1} y(n) e^{-j\frac{2\pi}{N}kn}, k = [0 \dots N-1] \ . \tag{19}$$

The DFT can also be expressed in matrix form. Just consider $y(n)$ and $Y(k)$ as elements of two N-component vectors $\mathbf{y}$ and $\mathbf{Y}$ related by

$$\mathbf{Y} = \mathbf{F}\mathbf{y} \ , \tag{20}$$

where $\mathbf{F}$ is the Fourier matrix, whose generic element of indices k, n is

$$F_{k,n} = e^{-j\frac{2\pi}{N}kn} \ . \tag{21}$$

It is clear that the sequence $\mathbf{y}$ can be recovered by premultiplication of the sequence $\mathbf{Y}$ by the matrix $\mathbf{F}^{-1}$, which is the inverse Fourier matrix. This can be expressed as

$$y(n) = \frac{1}{N} \sum_{k=0}^{N-1} Y(k) e^{j\frac{2\pi}{N}kn}, n = [0 \dots N-1] \ , \tag{22}$$

which is called the Inverse Discrete Fourier Transform.

The Fast Fourier Transform (FFT) [65, 67], is a fast algorithm for computing the sum (19). Namely, the FFT has computational complexity [24] of the order of $N \log N$, while the trivial procedure for computing the sum (19) would take an order of N^2 steps, thus being intractable in many practical cases. The FFT can be found as a predefined component in most systems for digital signal processing and sound processing languages. For instance, there is an `fft` builtin function in Octave, CSound, CLM (see the appendix B).

1.4 Discrete-Time Systems

A discrete-time system is any processing block that takes an input sequence of samples and produces an output sequence of samples. The actual processing

can be performed sample by sample or as a sequence of transformations of data blocks.

The linear and time-invariant systems are particularly interesting because a theory is available that describes them completely. Since we have already seen in sec. 1.1 what we mean by linearity, here we restate the concept with formulas. If $y_1(n)$ and $y_2(n)$ are the system responses to the inputs $x_1(n)$ and $x_2(n)$ then, feeding the system with the input

$$x(n) = a_1 x_1(n) + a_2 x_2(n) \tag{23}$$

we get, at each discrete instant n

$$y(n) = a_1 y_1(n) + a_2 y_2(n) \ . \tag{24}$$

In words, the superposition principle does hold.

The time invariance is defined by considering an input sequence $x(n)$, which gives an output sequence $y(n)$, and a version of $x(n)$ shifted by D samples: $x(n-D)$. If the system is time invariant, the response to $x(n-D)$ is equal to $y(n)$ shifted by D samples, i.e. $y(n-D)$. In other words, the time shift can be indifferently put before or after a time-invariant system. Cases where the time invariance does not hold are found in systems that change their functionality over time or that produce an output sequence at a rate different from that of the input sequence (e.g., a decimator that undersamples the input sequence).

An important property of linear and time-invariant (LTI) systems is that, in a cascade of LTI blocks the order of such blocks is irrelevant for the global input-output relation.

As we have already mentioned for continuous-time systems, there are two important system descriptions: the impulse response and the transfer function. LTI discrete-time systems are completely described by either one of these two representations.

1.4.1 The Impulse Response

Any input sequence can be expressed as a weighted sum of discrete impulses properly shifted in time. A discrete impulse is defined as

$$\delta(n) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} \ . \tag{25}$$

If the impulse (25) gives as output a sequence (called, indeed, the impulse response) $h(n)$ defined in the discrete domain, then a linear combination of shifted impulses will produce a linear combination of shifted impulse responses.

Therefore, it is easy to be convinced that the output can be expressed by the following general convolution[6]:

$$y(n) = (h * x)(n) = \sum_m x(m)h(n-m) = \sum_m h(m)x(n-m) \ , \qquad (26)$$

which is the discrete-time version of (6).

The Z transform $H(z)$ of the impulse response is called transfer function of the LTI discrete-time system. By analogy to what we showed in sec. 1.1, the input-output relationship for LTI systems can be described in the transform domain by

$$Y(z) = H(z)X(z) \ , \qquad (27)$$

where the input and output signals $X(z)$ and $Y(z)$ have been capitalized to indicate that these are the Z transforms of the signals themselves.

The following general rule can be given:

- A linear and time-invariant system working in continuous or discrete time can be represented by an operation of convolution in the time domain or, equivalently, by a complex multiplication in the (respectively Laplace or Z) transform domain. The results of the two operations are related by a (Laplace or Z) transform.

Since the transforms can be inverted the converse statement is also true:

- The convolution between two signals in the transform domain is the transform of a multiplication in the time domain between the antitransforms of the signals.

1.4.2 The Shift Theorem

We have seen how two domains related by a transform operation such as the Z transform are characterized by the fact that the convolution in one domain corresponds to the multiplication in the other domain. We are now interested to know what happens in one domain if in the other domain we perform a shift operation. This is stated in the

Theorem 1.2 (Shift Theorem) *Given two domains related by a transform operator, the shift by τ in one domain corresponds, in the transform domain, to a multiplication by the kernel of the transform raised to the power τ.*

[6]The reader is invited to construct an example with an impulse response that is different from zero only in a few points.

We recall that the kernel of the Laplace transform[7] is e^{-s} and the kernel of the Z transform is z^{-1}. The shift theorem can be easily justified in the discrete domain starting from the definition of Z transform. Let $x(n)$ be a discrete-time signal, and let $y(n)$ be its version shifted by an integer number τ of samples. With the variable substitution $N = n - \tau$ we can produce the following chain of identities, which proves the theorem:

$$\begin{aligned} Y(z) &= \sum_{n=-\infty}^{\infty} y(n) z^{-n} = \sum_{n=-\infty}^{\infty} x(n-\tau) z^{-n} = \\ &= \sum_{N=-\infty}^{\infty} x(N) z^{-N-\tau} = z^{-\tau} X(z) \; . \end{aligned} \tag{28}$$

1.4.3 Stability and Causality

The notion of causality is rather intuitive: it corresponds to the experience of exciting a system and getting its response back only in future time instants, i.e. in instants that follow the excitation time along the time arrow. It is easy to realize that, for an LTI system, causality is enforced by forbidding non-zero values to the impulse response for time instants preceding zero. Non-causal systems, even though not realizable by sample-by-sample processing, can be of interest for non-realtime applications or where a processing delay can be tolerated.

The notion of stability is more delicate and can be given in different ways. We define the so-called bounded-input bounded-output (BIBO) stability, which requires that any input bounded in amplitude might only produce a bounded output, even though the two bounds can be different. It can be shown that having BIBO stability is equivalent to have an impulse response that is absolutely summable, i.e.

$$\sum_{-\infty}^{\infty} |h(n)| < \infty \; . \tag{29}$$

In particular, a necessary condition for BIBO stability is that the impulse response converges toward zero for time instants diverging from zero.

It is easy to detect stability on the complex plane for LTI causal systems [58, 66, 65]. In the continuous-time case, the system is stable if all the poles are on the left of the imaginary axis or, equivalently, if the strip of convergence (see

[7] This is the kernel of the direct transform, being e^{s} the kernel of the inverse transform.

appendix A.8.1) ranges from a negative real number to infinity. In the discrete-time case, the system is stable if all the poles are within the unit circle or, equivalently, the ring of convergence (see appendix A.8.3) has the inner radius of magnitude less than one and the outer radius extending to infinity.

Stability is a condition that is almost always necessary for practical realizability of linear filters in computing systems. It is interesting to note that physical systems can be locally unstable but, in virtue of the principle of energy conservation, these instabilities must be compensated in other points of the systems themselves or of the other systems they are interacting with. However, in numeric implementations, even local instabilities can be a problem, since the numerical approximations introduced in the representations of variables can easily produce diverging signals that are difficult to control.

1.5 Continuous-time to discrete-time system conversion

In many applications, and in particular in sound synthesis by physical modeling, the design of a discrete-time system starts from the description of a physical continuous-time system by means of differential equations and constraints. This description of an analog system can itself be derived from the simplification of the physical reality into an assembly of basic mechanical elements, such as springs, dampers, frictions, nonlinearities, etc. . Alternatively, our continuous-time physical template can result from measurements on a real physical system. In any case, in order to construct a discrete-time system capable to reproduce the behavior of the continuous-time physical system, we need to transform the differential equations into difference equations, in such a way that the resulting model can be expressed as a signal flowchart in discrete time.

The techniques that are most widely used in signal processing to discretize a continuous-time LTI system are the impulse invariance and the bilinear transformation.

1.5.1 Impulse Invariance

In the method of the impulse invariance, the impulse response $h(n)$ of the discrete-time system is a uniform sampling of the impulse response $h_s(t)$ of the continuous-time system, rescaled by the width of the sampling interval T, according to

$$h(n) = T h_s(nT) \ . \tag{30}$$

In the usual practice of digital filter design, the constant T is usually neglected, since the design stems from specifications for the discrete-time filter, and the conversion to continuous time is only an intermediate stage. Since one should introduce $1/T$ when going from discrete to continuous time, and T when returning to discrete time, the overall effect of the constant is canceled. Vice versa, if we start from a description in continuous time, such as in physical modeling, the constant T should be considered.

From the sampling theorem we can easily deduce that the frequency response of the discrete-time system is the periodic replication of the frequency response of the continuous-time system, with a repetition period equal to $F_s = 1/T$. In terms of "discrete-time frequency" ω (in radians per sample), we can write

$$H(\omega) = \sum_{k=-\infty}^{\infty} H_s\left(\frac{j\omega}{T} + j\frac{2\pi}{T}k\right) = \sum_{k=-\infty}^{\infty} H_s\left(j\Omega + j2\pi F_s k\right) \ . \qquad (31)$$

The equation (31) shows that the frequency components in the two domains, discrete and continuous, can be identical in the base band only if the continuous-time system is bandlimited. If this is not the case (and it is almost never the case!), there will be some aliasing that introduces spurious components in the band of interest of the discrete-time system. However, if the frequency response of the continuous-time system is sufficiently close to zero in high frequency, the aliasing can be neglected and the resulting discrete-time system turns out to be a good approximation of the continuous-time template.

Often, the continuous-time impulse response is derived from a decomposition of the transfer function of a system into simple fractions. Namely, the transfer function of a continuous-time system can be decomposed[8] into a sum of terms such as

$$H_s(s) = \frac{a}{s - s_a} \ , \qquad (32)$$

which are given by impulse responses such as

$$h_s(t) = ae^{s_a t}1(t) \ , \qquad (33)$$

where $1(t)$ is the ideal step function, or Heaviside function, which is zero for negative (anticausal) time instants. Sampling the (33) we produce the discrete-time response

$$h(n) = Ta\left(e^{s_a T}\right)^n 1(n) \ , \qquad (34)$$

[8]This holds for simple distinct poles. The reader might try to extend the decomposition to the case of coincident double poles.

whose transfer function in z is

$$H(z) = \frac{Ta}{1 - e^{s_a T} z^{-1}} \ . \tag{35}$$

By comparing (35) and (32) it is clear what is the kind of operation that we should apply to the s-domain transfer function in order to obtain the z-domain transfer function relative to the impulse response sampled with period T.

It is important to recognize that the impulse-response method preserves the stability of the system, since each pole of the left s hemiplane is matched with a pole that stays within the unit circle of the z plane, and vice versa. However, this kind of transformation can not be considered a conformal mapping, since not all the points of the s plane are coupled to points of the z plane by a relation[9] $z = e^{sT}$. An important feature of the impulse-invariance method is that, being based on sampling, it is a linear transformation that preserves the shape of the frequency response of the continuous-time system, at least where aliasing can be neglected.

It is clear that the method of the impulse invariance can be used when the continuous-time reference model is a lowpass or a bandpass filter (see sec. 2 for a treatment of filters). If the template is an high-pass filtering block the method is not applicable because of aliasing.

1.5.2 Bilinear Transformation

An alternative approach to using the impulse invariance to discretize continuous systems is given by the bilinear transformation, a conformal map that creates a correspondence between the imaginary axis of the s plane and the unit circumference of the z plane. A general formulation of the bilinear transformation is

$$s = h \frac{1 - z^{-1}}{1 + z^{-1}} \ . \tag{36}$$

It is clear from (36) that the dc component $j0$ of the continuous-time system corresponds to the dc component $1 + j0$ of the discrete-time system, and the infinity of the imaginary axis of the s plane corresponds to the point $-1 + j0$, which represents the Nyquist frequency in the z plane. The parameter h allows to impose the correspondence in a third point of the imaginary axis of

[9] To be convinced of that, consider a second order continuous-time transfer function with simple poles and a zero and convert it with the method of the impulse invariance. Verify that the zero does not follow the same transformation that the poles are subject to.

the s plane, thus controlling the compression of the axis itself when it gets transformed into the unit circumference.

A particular choice of the parameter h derives from the numerical integration of differential equations by the trapezoid rule. To understand this point, consider the transfer function (32) and its relative differential equation that couples the input variable x_s to the output variable y_s

$$\frac{dy_s(t)}{dt} - s_a y_s(t) = a x_s(t) \; . \tag{37}$$

If we sample the output variable with period T we can write

$$y_s(nT) = y_s(nT - T) + \int_{nT-T}^{nT} \dot{y}_s(\tau) d\tau \; , \tag{38}$$

where $\dot{y}_s = \frac{dy_s(t)}{dt}$, and integrate the (38) with the trapezoid rule, thus obtaining

$$y_s(nT) \approx y_s((n-1)T) + (\dot{y}_s(nT) + \dot{y}_s((n-1)T))\, T/2 \; . \tag{39}$$

By replacing (37) into (39) and setting $y(n) = y_s(nT)$ we get a difference equation represented, in virtue of the shift theorem 1.2, by the transfer function

$$H(z) = \frac{a(1 + z^{-1})T/2}{1 - s_a T/2 - (1 + s_a T/2) z^{-1}} \; , \tag{40}$$

which can be obviously obtained from $H_s(s)$ by bilinear transformation with $h = 2/T$.

It is easy to check that, with $h = \frac{2}{T}$, the continuos-time frequency $f = \frac{1}{\pi T}$ maps into the discrete-time frequency $\omega = \frac{\pi}{2}$, i.e. half the Nyquist limit. More generally, half the Nyquist frequency of the discrete-time system corresponds to the frequency $f = \frac{h}{2\pi}$ of the continuous-time system. The more h is high, the more the low frequencies are compressed by the transformation.

To give a practical example, using the sampling frequency $F_s = 44100$Hz and $h = \frac{2}{T} = 88200$, the frequency that is mapped into half the Nyquist rate of the discrete-time system (i.e., 11025Hz), is $f = 14037.5$Hz. The same transformation, with $h = 100000$ maps the frequency $f = 15915.5$Hz to half the Nyquist rate. If we are interested in preserving the magnitude and phase response at $f = 11025$Hz we need to use $h = 69272.12$.

1.6 Quantization

With the adjectives "numeric" and "digital" we connote systems working on signals that are represented by numbers coded according to the conventions of appendix A.9. So far, in this chapter we have described discrete-time systems by means of signals that are functions of a discrete variable and having a codomain described by a continuous variable. Actually, the internal arithmetic of computing systems imposes a signal quantization, which can produce various kinds of effects on the output sounds.

For the scope of this book the most interesting quantization is the linear quantization introduced, for instance, in the process of conversion of an analog signal into a digital signal. If the word representing numerical data is b bits long, the range of variation of the analog signal can be divided into 2^b quantization levels. Any signal amplitude between two quantization levels can be quantized to the closest level. The processes of sampling and quantization are illustrated in fig. 4 for a wordlength of 3 bits. The minimal amplitude difference that can be represented is called the quantum interval and we indicate it with the symbol q. We can notice from fig. 4 that, due to two's complement representation, the representation levels for negative amplitude exceed by one the levels used for positive amplitude. It is also evident from fig. 4 how quant-

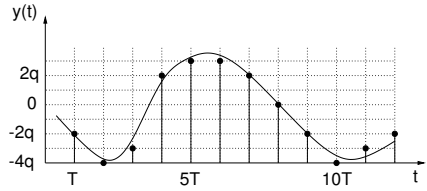

Figure 4: Sampling and 3-bit quantization of a continuous-time signal

ization introduces an approximation in the representation of a discrete-time signal. This approximation is called quantization error and can be expressed as

$$\eta(n) \stackrel{\triangle}{=} y_q(n) - y(n) \ , \tag{41}$$

where the symbol $y_q(n)$ indicates the value $y(n)$ quantized by rounding it to the nearest discrete level. From the viewpoint of the designer, the quantization noise can be considered as a noise superimposed to the unquantized signal.

This noise takes values in the range

$$-\frac{q}{2} \leq \eta \leq \frac{q}{2} \,, \tag{42}$$

and it is spectrally colored according to the nature and form of the unquantized signal.

What follows is a superficial analysis of quantization noises. In order to do a rigorous analysis we should assume that the reader has a background in random variables and processes. We rather refer to signal processing books [58, 67, 65] for a more accurate exposition.

In order to study the effects of quantization noise analytically, it is often assumed that it is a white noise (i.e., a noise with a constant-magnitude spectrum) with values uniformly distributed in the interval (42), and that there is no correlation between the noise and the unquantized signal. This assumption is false in general but, nevertheless, it leads to results which are good estimates of many actual behaviors. The uniformly-distributed white noise has a zero mean but it has a nonzero quadratic mean (i.e., a power) with value

$$\overline{\eta^2} = \frac{1}{q/2} \int_0^{q/2} \eta^2 d\eta = \frac{q^2}{12} \,. \tag{43}$$

In the frequency domain, the quantization noise is interpreted by means of a spectrum such as that depicted in fig. 5, which represents the square of the magnitude of the Fourier transform. The area of the dashed rectangle is equal to the power $\overline{\eta^2}$. Usually the root-mean-square value (or RMS value) of the

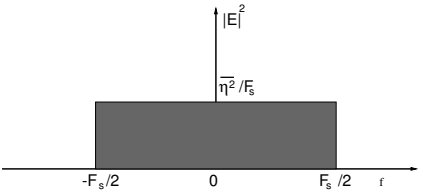

Figure 5: Squared magnitude spectrum of an ideal quantization noise

quantization noise is given, and this is defined as

$$\eta_{rms} = \sqrt{\overline{\eta^2}} = \frac{q}{\sqrt{12}} \,, \tag{44}$$

which can be directly compared with the maximal representable value in order to get the signal-to-quantization noise ratio (or SNR)

$$SNR = 20\log_{10}\frac{q2^{b-1}}{q/\sqrt{12}} = 20\log_{10}(2^b\sqrt{3}) \approx 4.7 + 6\,b\,\text{dB}\ . \qquad (45)$$

As a general rule, each further quantization bit increases the SNR by 6dB. Therefore, with 16 bits we have a signal-to-quantization noise ratio of about 101.1dB. When we are given a SNR of 96.3dB with 16 bits, it means that the ratio has been computed using the maximum value $q/2$ of the quantization noise and not its RMS value, which is more significant for the human ear. The definition (45) is that proposed by Steiglitz [102].

The assumptions on the statistical properties of the quantization noise are better verified if the signal is large in amplitude and wide in its frequency extension. For quasi-sinusoidal signals the quantization noise is heavily colored and correlated with the unquantized signal, in such an extent that some additive noise called dither is sometimes introduced in order to whiten and decorrelate the quantization noise. In this way, the perceptual effects of quantization turn out to be less severe.

By considering the quantization noise as an additive signal we can easily study its effects within linear systems. The operations performed by a discrete-time linear system, especially when done in fixed-point arithmetics, can indeed modify the spectral content of noise signals, and different realizations of the same transfer functions can behave very differently as far as their immunity to quantization noise is concerned. Several quantizations can occur within the realization of a linear system. For instance, the multiplication of two fixed-point numbers represented with b bits requires $2b-1$ bits to represent the result without any precision loss. If successive operations use operands represented with b bits it is clear that the least-significant bits must be eliminated, thus introducing a quantization. The effects of these quantizations can be studied resorting to the additive white noise model, where the points of injection of noises are the points where the quantization actually occurs.

The fixed-point implementations of linear systems are subject to disappointing phenomena related to quantization: limit cycles and overflow oscillations. Both phenomena can be expressed as nonzero signals that are maintained even when the system has stopped to produce usuful signals. The limit cycles are usually small oscillations due to the fact that, because of rounding, the sources of quantization noise determine a local amplification or attenuation of the signal (see fig. 4). If the signals within the system have a physical meaning (e.g., they are propagating waves), the limit cycles can be avoided by

forcing a lossy quantization, which truncates the numbers always toward zero. This operation corresponds to introducing a small numerical dissipation. The overflow oscillations are more serious because they produce signals as large as the maximum amplitude that can be represented. They can be produced by operations whose results exceed the largest representable number, so that the result is slapped back into the legal range of two's complement numbers. Such a distructive oscillation can be avoided by using overflow-protected operations, which are operations that saturate the result to the largest representable number (or to the most negative representable number).

The quantizations introduce nonlinear elements within otherwise linear structures. Indeed, limit cycles and overflow oscillations can persist only because there are nonlinearities, since any linear and stable system can not give a persistent nonzero output with a zero input.

Quantization in floating point implementations is usually less of a concern for the designer. In this case, quantization occurs only in the mantissa. Therefore, the relative error

$$\eta_r(n) \triangleq \frac{y_q(n) - y(n)}{y(n)} \ , \tag{46}$$

is more meaningful for the analysis. We refer to [65] for a discussion on the effects of quantization with floating point implementations.

Some digital audio formats, such as the μ-law and A-law encodings, use a fixed-point representation where the quantization levels are distributed non linearly in the amplitude range. The idea, resemblant of the quasi logarithmic sensitivity of the ear, is to have many more levels where signals are small and a coarser quantization for large amplitudes. This is justified if the signals being quantized do not have a statistical uniform distribution but tend to assume small amplitudes more often than large amplitudes. Usually the distribution of levels is exponential, in such a way that the intervals between points increase exponentially with magnitude. This kind of quantization is called logarithmic because, in practical realizations, a logarithmic compressor precedes a linear quantization stage [69]. Floating-point quantization can be considered as a piecewise-linear logarithmic quantization, where each linear piece corresponds to a value of the exponent.

Chapter 2

Digital Filters

For the purpose of this book we call digital filter any linear, time-invariant system operating on discrete-time signals. As we saw in chapter 1, such a system is completely described by its impulse response or by its (rational) transfer function. Even though the adjective digital refers to the fact that parameters and signals are quantized, we will not be too concerned about the effects of quantization, that have been briefly introduced in sec. 1.6. In this chapter, we will face the problem of designing impulse responses or transfer functions that satisfy some specifications in the time or frequency domain.

Traditionally, digital filters have been classified into two large families: those whose transfer function doesn't have the denominator, and those whose transfer function have the denominator. Since the filters of the first family admit a realization where the output is a linear combination of a finite number of input samples, they are sometimes called non-recursive filters[1]. For these systems, it is more customary and correct to refer to the impulse response, which has a finite number of non-null samples, thus calling them Finite Impulse Response (FIR) filters. On the other hand, the filters of the second family admit only recursive realizations, thus meaning that the output signal is always computed by using previous samples of itself. The impulse response of these filters is infinitely long, thus justifying their name as Infinite Impulse Response (IIR) filters.

[1] Strictly speaking, this definition is not correct because the same transfer functions can be realized in recursive form

2.1 FIR Filters

An FIR filter is nothing more than a linear combination of a finite number of samples of the input signal. In our examples we will treat causal filters, therefore we will not process input samples coming later than the time instant of the output sample that we are producing.

The mathematical expression of an FIR filter is

$$y(n) = \sum_{m=0}^{N} h(m)x(n-m) \ . \tag{1}$$

In eq. 1 the reader can easily recognize the convolution (26), here specialized to finite-length impulse responses. Since the time extension of the impulse response is $N+1$ samples, we say that the FIR filter has length $N+1$.

The transfer function is obtained as the Z transform of the impulse response and it is a polynomial in the powers of z^{-1}:

$$H(z) = \sum_{m=0}^{N} h(m)z^{-m} = h(0) + h(1)z^{-1} + \cdots + h(N)z^{-N} \ . \tag{2}$$

Since such polynomial has order N, we also say that the FIR filter has order N.

2.1.1 The Simplest FIR Filter

Let us now consider the simplest nontrivial FIR filter that one can imagine, the averaging filter

$$y(n) = \frac{1}{2}x(n) + \frac{1}{2}x(n-1) \ . \tag{3}$$

In appendix B.1 it is illustrated how such filter can be implemented in Octave/Matlab in two different ways: block processing or sample-by-sample processing. The simplest way to analyze the behavior of the filter [97] is probably the injection of a complex sinusoid having amplitude A and initial phase ϕ, i.e. the signal $x(n) = Ae^{j(\omega_0 n+\phi)}$. Since the system is linear we do not loose any generality by considering unit-amplitude signals ($A = 1$). Since the system is time invariant we do not loose any generality by considering signals with initial zero phase ($\phi = 0$). Since the complex sinusoid can be expressed as the sum of a cosinusoidal real part and a sinusoidal imaginary part, we can imagine that feeding the system with such a complex signal corresponds to feeding

two copies of the filter, the one with a cosinusoidal real signal, the other with a sinusoidal real signal. The output of the filter fed with the complex sinusoid is obtained, thanks to linearity, as the sum of the outputs of the two copies.

If we replace the complex sinusoidal input in eq. (3) we readily get

$$\begin{aligned} y(n) &= \frac{1}{2}e^{j\omega_0 n} + \frac{1}{2}e^{j\omega_0(n-1)} = (\frac{1}{2} + \frac{1}{2}e^{-j\omega_0})e^{j\omega_0 n} = \\ &= (\frac{1}{2} + \frac{1}{2}e^{-j\omega_0})x(n) \ . \end{aligned} \tag{4}$$

We see that the output is a copy of the input multiplied by the complex number $(\frac{1}{2} + \frac{1}{2}e^{-j\omega_0})$, wich is the value taken by the transfer function at the point $z = e^{j\omega_0}$. In fact, the transfer function (2) can be rewritten, for the case under analysis, as

$$H(z) = \frac{1}{2} + \frac{1}{2}z^{-1} \ , \tag{5}$$

and its evaluation on the unit circle ($z = e^{j\omega}$) gives the frequency response

$$H(\omega) = \frac{1}{2} + \frac{1}{2}e^{-j\omega} \ . \tag{6}$$

For an input complex sinusoid having frequency ω_0, the frequency response takes value

$$\begin{aligned} H(\omega_0) &= \frac{1}{2} + \frac{1}{2}e^{-j\omega_0} = (\frac{1}{2}e^{j\omega_0/2} + \frac{1}{2}e^{-j\omega_0/2})e^{-j\omega_0/2} = \\ &= \cos(\omega_0/2)e^{-j\omega_0/2} \ , \end{aligned} \tag{7}$$

and we see that the magnitude response and the phase response are, respectively

$$|H(\omega_0)| = \cos(\omega_0/2) \tag{8}$$

and

$$\angle H(\omega_0) = -\omega_0/2 \ . \tag{9}$$

These are respectively the magnitude and argument of the complex number that is multiplied by the input function in (4). Therefore, we have verified a general property of linear and time-invariant systems, i.e., sinusoidal inputs give sinusoidal outputs, possibly with an amplitude rescaling and a phase shift[2].

[2]The reader can easily verify that this is true not only for complex sinusoids, but also for real sinusoids. The real sinusoid can be expressed as a combination of complex sinusoids and linearity can be applied.

If the frequency of the input sine is thought of as a real variable ω in the interval $[0, \pi)$, the magnitude and phase responses become a function of such variable and can be plotted as in fig. 1. At this point, the interpretation of such curves as amplification and phase shift of sinusoidal inputs should be obvious.

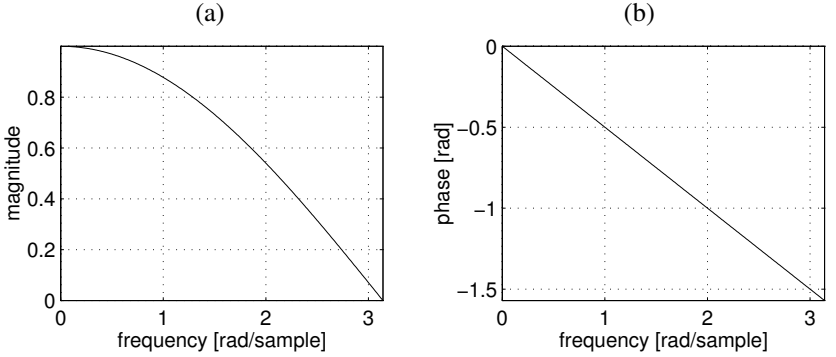

Figure 1: Frequency response (magnitude and phase) of an averaging filter

In order to plot curves such as those of fig. 1 it is not necessary to calculate closed forms of the functions representing the magnitude (8) and the phase response (9). Since with Octave/Matlab we can directly operate on arrays of complex numbers, the following simple script will do the job:

```
global_decl; platform('octave');
w = [0:0.01:pi];                    % frequency points
H = 0.5 + 0.5*exp(- i * w );        % complex freq. resp.
subplot(2,2,1); plot(w, abs(H));    % plot the magnitude
xlabel('frequency [rad/sample]');
ylabel('magnitude');
eval(myreplot);
subplot(2,2,2); plot(w, angle(H)); % plot the phase
xlabel('frequency [rad/sample]');
ylabel('phase [rad]');
eval(myreplot);
```

The averaging filter is the simplest form of lowpass filter. In a lowpass filter the high frequencies are more attenuated than the low frequencies. Another way to approach the analysis of a filter is to reason directly in the plane of the complex variable z. In this plane (fig. 2) two families of points are marked: the

points where the transfer function vanishes, and the points where it diverges to infinity. Let us rewrite the transfer function as the ratio of two polynomials in z

$$H(z) = \frac{1}{2}\frac{z - z_0}{z} \ , \tag{10}$$

where $z_0 = -1$ is the root of the numerator. The roots of the numerator of a transfer function are called zeros of the filter, and the roots of the denominator are called poles of the filter. Usually, for reasons that will emerge in the following, only the nonzero roots are counted as poles or zeros. Therefore, in the example (10) we have only one zero and no pole.

In order to evaluate the frequency response of the filter it is sufficient to replace the variable z with $e^{j\omega}$ and to consider $e^{j\omega}$ as a geometric vector whose head moves along the unit circle. The difference between this vector and the vector z_0 gives the cord drawn in fig. 2. The cord length doubles[3] the magnitude response of the filter. Such a chord, interpreted as a vector with the head in $e^{j\omega}$, has an angle that can be subtracted from the vector angle of the pole at the origin, thus giving the phase response of the filter at the frequency ω.

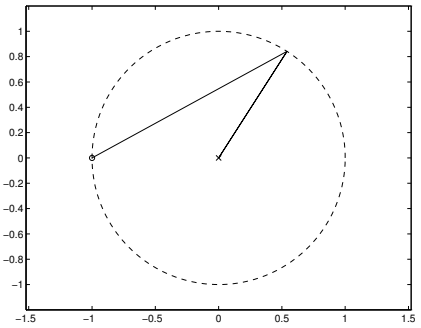

Figure 2: Single zero (○) and pole in the origin (×)

The following general rules can be given, for any number of poles and zeros:

- Considered a point $e^{j\omega}$ on the unit circle, the magnitude of the frequency response (regardless of constant factors) at the frequency ω is obtained by multiplication of the magnitudes of the vectors linking the zeros with

[3] Do not forget the scaling factor $\frac{1}{2}$ in (10).

the point $e^{j\omega}$, divided by the magnitudes of the vectors linking the poles with the point $e^{j\omega}$.

- The phase response is obtained by addition of the phases of the vectors linking the zeros with the point $e^{j\omega}$, and by subtraction of the phases of the vectors linking the poles with the point $e^{j\omega}$.

It is readily seen that poles or zeros in the origin do only contribute to the phase of the frequency response, and this is the reason for their exclusion from the total count of poles and zeros.

The graphic method, based on pole and zero placement on the complex plane is very useful to have a rough idea of the frequency response. For instance, the reader is invited to reconstruct fig. 1 qualitatively using the graphic method.

The frequency response gives a clear picture of the behavior of a filter when its inputs are stationary signals, which can be decomposed as constant-amplitude sinusoids. Therefore, the frequency response represents the steady-state response of the system. In practice, even signals composed by sinusoids have to be turned on at a certain instant, thus producing a transient response that comes before the steady-state. However, the knowledge of the Z transform of a causal complex sinusoid and the knowledge of the filter transfer function allow us to study the overall response analytically. As we show in appendix A.8.3, the Z transform of causal exponential sequence is

$$X(z) = \frac{1}{1 - e^{j\omega_0} z^{-1}} \; . \tag{11}$$

If we multiply, in the z domain, $X(z)$ by the transfer function $H(z)$ we get

$$\begin{aligned} Y(z) &= H(z)X(z) = \frac{1}{2}(1 + z^{-1}) \frac{1}{1 - e^{j\omega_0} z^{-1}} = \\ &= \frac{1}{2} \frac{1}{1 - e^{j\omega_0} z^{-1}} + \frac{1}{2} \frac{z^{-1}}{1 - e^{j\omega_0} z^{-1}} \; . \end{aligned} \tag{12}$$

The second term of the last member of (12) is, by the shift theorem, the transform of a causal complex sinusoidal sequence delayed by one sample. Therefore, the overall response can be thought of as a sum of two identical sinusoids shifted by one sample and this turns out to be another sinusoid, but only after the first sampling instant. The first instant has a different behavior since it is part of the transient of the response (see fig. 3). It is easy to realize that, for an FIR filter, the transient lasts for a number of samples that doesn't exceed the order (memory) of the filter itself. Since an order-N FIR filter has a memory of N samples, the transient is at most N samples long.

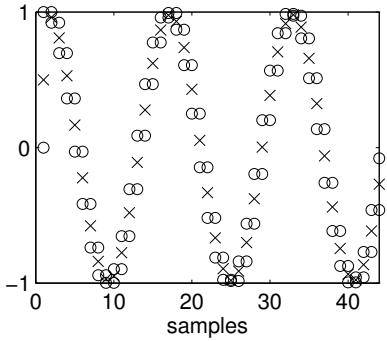

Figure 3: Response of an FIR averaging filter to a causal cosine: input and delayed input (○), actual response (×)

2.1.2 The Phase Response

If we filter a sound with a nonlinear-phase filter we alter its time-domain wave shape. This happens because the different frequency components are subject to a different delay while being transferred from the input to the output of the filter. Therefore, a compact wavefront is dispersed during its traversal of the filter. Before defining this concept more precisely we illustrate what happens to the wave shape that is impressed by a hammer to the string in the piano. The string behaves like a nonlinear-phase filter, and the dispersion of the frequency components becomes increasingly more evident while the wave shape propagates away from the hammer along the string. Fig. 4 illustrates the string displacement signal as it is produced by a physical model (see chapter 5 for details) of the hammer-string system. The initial wave shape progressively loses its initial form. In particular, the fact that high frequencies are subject to a smaller propagation delay than low frequencies is visible in the form of little precursors, i.e., small high-frequency oscillations that precede the return of the main components of the wave shape. Such an effect can be experienced with an aerial ropeway like those that are found in isolated mountain houses. If we shake the rope energetically and keep our hand on it, after a few seconds we perceive small oscillations preceding a strong echo.

The effects of the phase response of a filter can be better formalized by introducing two mathematical definitions: the phase delay and the group delay.

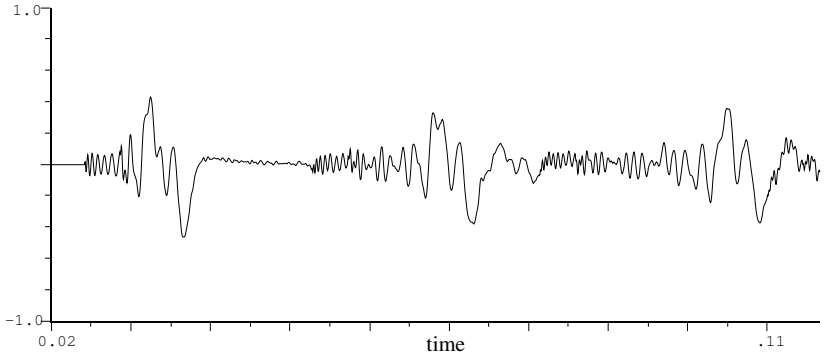

Figure 4: Struck string: string displacement at the bridge termination

The phase delay is defined as

$$\tau_{ph} \triangleq -\frac{\angle H(\omega)}{\omega} , \tag{13}$$

i.e., at any frequency, it is given by the phase response divided by the frequency itself. In practice, given the phase-response curve, the phase delay at one point is obtained as the slope of the straight line that connects that point with the origin. The group delay is defined in differential terms as

$$\tau_{gr} \triangleq -\frac{d\angle H(\omega)}{d\omega} . \tag{14}$$

Therefore, the group delay at one point of the phase-response curve, is equal to the slope of the curve. The fig. 5 illustrates the difference between phase delay and group delay. It is clear that, if the phase is linear, the two delays are equal and coincident with the slope of the straight line that represents the phase response.

The difference between local slope and slope to the origin is crucial to understand the physical meaning of the two delays. The phase delay at a certain frequency point is the delay that a single frequency component is subject to when it passes through the filter, and the quantity (13) is, indeed, a delay in samples. Vice versa, in order to interpret the group delay let us consider a local approximation of the phase response by the tangent line at one point. Locally, propagation can be considered linear and, therefore, a signal having frequency components focused around that point has a time-domain envelope

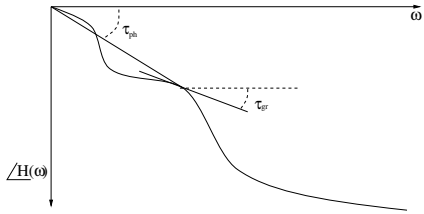

Figure 5: Phase delay and group delay

that is delayed by an amount proportional to the slope of the tangent. For instance, two sinusoids at slightly different frequencies are subject to beats and the beat frequency is the difference of the frequency components (see fig. 6). Therefore, beats are a frequency local phenomenon, only dependent on the relative distance between the components rather than on their absolute positions. If we are interested in knowing how the beat pattern is delayed by a filter, we should consider local variations in the phase curve. In other words, we should consider the group delay.

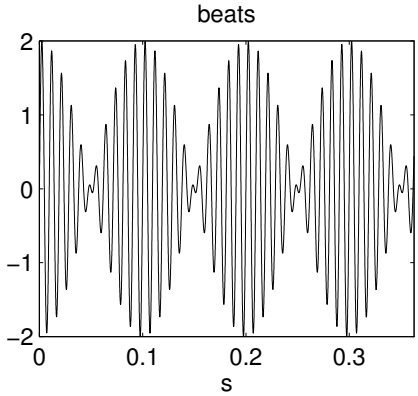

Figure 6: Beats between a sine wave at 100 Hz and a sine wave at 110 Hz

In telecommunications the group delay is often the most significant between the two delays, since messages are sent via wave packets localized in a narrow frequency band, and preservation of the shape of such packets is important. Vice versa, in sound processing it is more meaningful to consider the set of

frequency components in the audio range as a whole, and the phase delay is more significant. In both cases, we have to be careful of a problem that often arises when dealing with phases: the phase unwrapping. So far we have defined the phase response as the angle of the frequency response, without bothering about the fact that such an angle is defined univocally only between 0 and 2π. There is no way to distinguish an angle θ from those angles obtained by addition of θ with multiples of 2π. However, in order to give continuity to the phase and group delays, we have to unwrap the phase into a continuous function. For instance, the Matlab Signal Processing Toolbox provides the function `unwrap` that unwraps the phase in such a way that discontinuities larger than a given threshold are offset by 2π. In Octave we can use the function `unwrap` found in the web repository of this book.

Example 1. Fig. 7 shows the phase response of the FIR filter $H(z) = 0.5 - 0.2z^{-1} - 0.3z^{-2} + 0.8z^{-3}$ before and after unwrapping. The following Octave/Matlab script allows to plot the curve in fig. 7. It is illustrative of the usage of the function `unwrap` with the default unwrapping threshold set to π.

```
w = [0:0.01:pi];
H = 0.5 - 0.2*exp(-i*w ) - 0.3*exp(-2*i*w ) + \
    0.8*exp(-3*i*w ) ;
plot(w, unwrap(angle(H)), '-'); hold on;
plot(w, angle(H), '--'); hold off;
xlabel('frequency [rad/sample]');
ylabel('phase [rad]');
title('Phase response');
% replot; % Octave only
```

###

2.1.3 Higher-Order FIR Filters

An FIR filter is nothing more than the realization of the operation of convolution (1). The filter coefficients are the samples of the impulse response.

The FIR filters having an impulse response that is symmetric are particularly important, since the phase of their frequency response is linear. More precisely, a symmetric impulse response is such that

$$h(n) = h(N - n),\ n = [0, \ldots, N]\ , \tag{15}$$

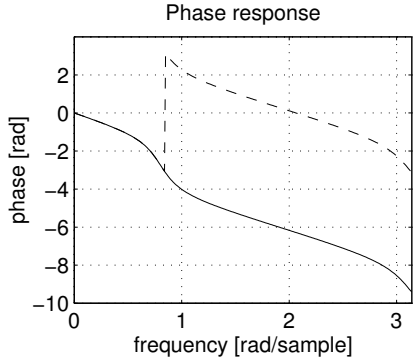

Figure 7: Wrapped (dashed line) and unwrapped (solid line) phase response of a third order FIR filter having impulse response: 0.5 -0.2 -0.3 0.8

and an antisymmetric impulse response is such that

$$h(n) = -h(N-n),\ n = [0, \ldots, N]\ . \tag{16}$$

It is possible to show that the symmetry (or antisymmetry) of the impulse response is a sufficient condition to ensure the linearity of phase[4]. This property is important to ensure the invariance of the shape of signals going through the filter. For instance, if a sawtooth signal is the input of a linear-phase lowpass filter, the output is still a sawtooth signal with rounded corners.

In order to prove that symmetry is a sufficient condition for phase linearity for an N-th order FIR filter (with N odd integer), we write the transfer function as

$$\begin{aligned} H(z) &= h(0) + \cdots + h(\frac{N-1}{2}) z^{-\frac{N-1}{2}} + \\ &\quad + h(\frac{N-1}{2}) z^{-\frac{N+1}{2}} + \cdots + h(0) z^{-N} \\ &= \sum_{n=0}^{\frac{N-1}{2}} h(n) \left(z^{-n} + z^{-N+n} \right)\ . \end{aligned} \tag{17}$$

[4]Actually, for antisymmetric odd-order filters, linear phase is achieved if $h(\frac{N-1}{2}) = 0$

The frequency response can be expressed as

$$\begin{aligned} H(\omega) &= \sum_{n=0}^{\frac{N-1}{2}} h(n) \left(e^{-j\omega n} + e^{j\omega(-N+n)} \right) \\ &= \sum_{n=0}^{\frac{N-1}{2}} h(n) e^{-j\omega \frac{N}{2}} \left(e^{-j\omega(n-\frac{N}{2})} + e^{j\omega(n-\frac{N}{2})} \right) \\ &= e^{-j\omega \frac{N}{2}} 2 \sum_{n=0}^{\frac{N-1}{2}} h(n) \cos\left(\omega(n - \frac{N}{2})\right) . \end{aligned} \tag{18}$$

In the latter term we have isolated the phase contribution from a (real) weighted sum of sinusoidal functions. The phase contribution is a straight line having slope $-N/2$, as we have already seen in the special case of the first-order averaging filter (5). Where the real term changes sign there are indeed 180° phase shifts, so that we should more precisely say that the phase is piecewise linear. However, phase discontinuities at isolated points do not alter the overall constancy of group delay, and they are nevertheless irrelevant because at those points the magnitude is zero.

The same property of piecewise phase linearity holds for antisymmetric impulse responses and for even values of N.

At this point, we are going to introduce a very useful FIR filter. It is linear phase and it has order 2 (i.e., length 3). The averaging filter (5) was also a linear phase filter, but it is not possible to change the shape of its frequency response without giving up the phase linearity. In fact, filters having form $H(z) = h(0) + h(1)z^{-1}$ can have linear phase only if $h(0) = \pm h(1)$, and this force them to have a magnitude response such as that of fig. 1 or like its high-pass mirrored version[5]. The filter that we are going to analyze has transfer function

$$H(z) = a_0 + a_1 z^{-1} + a_0 z^{-2} . \tag{19}$$

The impulse response is symmetric and, therefore, its phase response is linear. The frequency response can be calculated as

$$\begin{aligned} H(\omega) &= a_0 + a_1 e^{-j\omega} + a_0 e^{-2j\omega} \\ &= e^{-j\omega} \left(a_0 e^{j\omega} + a_1 + a_0 e^{-j\omega} \right) \\ &= e^{-j\omega} \left(a_1 + 2a_0 \cos\omega \right) . \end{aligned} \tag{20}$$

[5]The reader can analyze the filter $H(z) = 0.5 - 0.5z^{-1}$ and verify that it is a highpass filter.

As we have anticipated, the phase is linear and we have a phase delay of one sample. The magnitude of the frequency response is a function of the two parameters a_0 and a_1. Therefore, the designer has two degrees of freedom to control, for instance, the magnitude of the frequency response at two distinct frequencies.

A first property that one might want to impose is a lowpass shape to the frequency response. The reader, starting from (20), can easily verify that a sufficient condition to ensure that the magnitude of the frequency response is a decreasing monotonic function is that

$$a_1 \geq 2a_0 \geq 0 \; . \tag{21}$$

If we want to set the magnitude A_1 at the frequency ω_1 and the magnitude A_2 at the frequency ω_2 we have to solve the linear system of equations

$$\begin{aligned} a_1 + 2a_0 \cos\omega_1 &= A1 \\ a_1 + 2a_0 \cos\omega_2 &= A2 \; , \end{aligned}$$

that can be expressed in matrix form as

$$\begin{bmatrix} 1 & 2\cos\omega_1 \\ 1 & 2\cos\omega_2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \; . \tag{22}$$

For instance, if $\omega_1 = 0.01, \omega_2 = 2.0, A_1 = 1.0$ and $A_2 = 0.5$, in Octave/Matlab a system such as this can be written and solved with the script

```
w1 = 0.01; w2 = 2.0;
A1 = 1.0; A2 = 0.5;
A = [ 1 2*cos(w1) ; 1 2*cos(w2)];
b = [A1 ; A2];
a = A \ b;  % solution of the system b = A a
```

and the solutions returned for the variables a_1 and a_0 are, respectively,

```
a=
  0.64693
  0.17654
```

The frequency response of this filter is shown in fig. 8. If we design the second-order filter by specification of the frequency response at two arbitrary frequencies, we can easily get a magnitude response larger than one at zero frequency (also called dc frequency). Especially in signal processing flowgraphs

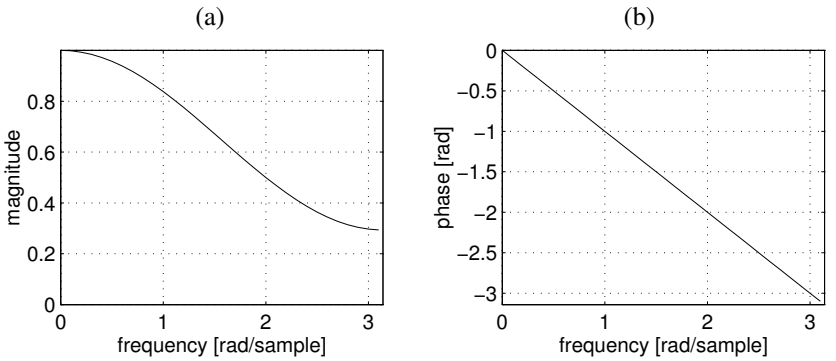

Figure 8: Frequency response (magnitude (a) and phase (b)) of the length-3 linear phase FIR filter with coefficients $a_0 = 0.17654$ and $a_1 = 0.64693$

having loops it is often desirable to normalize the maximum value of the magnitude response to one, in such a way that amplifications generating instabilities can be avoided. Of course, it is always possible to rescale the filter input or output by a scalar that is reciprocal to $H(0) = a_1 + 2a_0$ so that the response is forced to be unitary at dc[6]. Instead of drawing the pole-zero diagram of the filter, let us represent the contours of the logarithm of the magnitude of the transfer function, evaluated on the complex plane in a square centered on the origin (see fig. 9). The effects of the double pole in the origin and of the zeros $z = -0.29695$ and $z = -3.36754$ are clearly visible. A filter such as (8) has been proposed as part of an algorithm for synthesis of plucked string sounds [104].

We have seen that an FIR filter is the realization of a convolution between the input signal and the sequence of coefficients. The computation of this convolution can be made explicit in a language such as Octave and, indeed, this is what we have done in the appendix B.1 for the simple filter of length 2. For high-order filters it is more convenient to use algorithms that increase the efficiency of convolution. In Octave, there is the function `fftfilt` that, given a vector `b` of coefficients and an input signal `x`, returns the output of the FIR filter[7]. In order to perform this computation, the `fftfilt` computes an FFT

[6]The reader is invited to reformulate the system (22) with $\omega_1 = 0$ and $\omega_2 = \pi$. This corresponds to setting the magnitude at dc and Nyquist rate.

[7]In Matlab, the same function is available in the Signal Processing Toolbox. In any case, the Octave version `fftfilt`, avaliable in the web repository of this book, can also be used in Matlab.

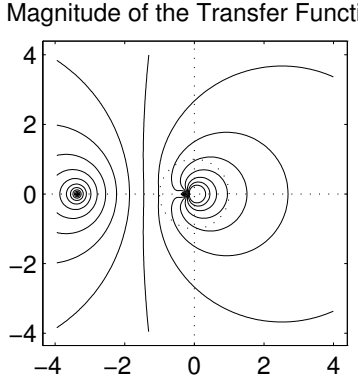

Figure 9: Magnitude of the transfer function [in dB] of an order-2 FIR filter on the complex z plane

of the coefficients and an FFT of the input signal, it multiplies the two transforms point by point (convolution in the time domain is multiplication in the transform domain), and it applies an inverse FFT to the result. Since the FFT of a length-N sequence has complexity of the order of $N \log N$ and the point-by-point multiply has complexity of the order of N, the convolution computed in this way has complexity of the order of $N \log N$. For sequences longer than a few samples, such a procedure is much faster than direct convolution. For even longer sequences, it is convenient to decompose the sequences into blocks and repeat the operations block by block. The partial results are then recomposed by partial addition of neighboring blocks of results. The detailed explanation of this technique is reported in several signal processing books, such as [67].

Most sound processing languages and real-time sound processing environments have primitive functions to compute the output of FIR filters. For instance, in SAOL (see appendix B.2) there is the function `fir(input, h0, h1, h2, ...)` that takes the input signal and the filter coefficients as arguments.

Example 2. In order to strengthen our understanding of FIR filters, we approach the design of a 10-th order linear phase filter having unit response at dc and an attenuation of 20dB at $F_s/6$. The impulse response of a 10-th order (or length 11) filter can be considered as the convolution of the responses of 5 2-nd order filters. Therefore, it is sufficient to design a length-3 filter with

a slighter attenuation at $F_s/6$ and to convolve five copies of this filter. The reader is invited to design the filter and to experience its effect using a sound processing language or real-time environment. A related task is the design of a highpass filter of the same length having a magnitude response that is symmetric to the response of the lowpass filter. Is there any law of symmetry that relates the coefficients of the two filters? How are the zeros distributed in the complex plane in the two cases? A further interesting exercise is the analysis and experimentation of the frequency response of the parallel connection of the two filters.

Development. The Octave/Matlab script that follows answers most of the questions. The remaining questions are left to the reader.

```
global_decl;
plat = platform('octave');
w0=0; A0=1;             % Response at dc
w1=pi/3; A1=0.1^(1/5); % Response at Fs/6 (1/5 of 20 dB)
%% coefficients of the length-3 FIR filter
A = [1 2*cos(w0); 1 2*cos(w1)]; b = [A0; A1];
a = A\b;
a1 = a(1)
a0 = a(2)
w = [0:0.01:pi];
%% frequency response of the length-3 FIR filter
H = a0 + a1*exp(-i*w) + a0*exp(-i*2*w);
%% frequency response of the length-11 FIR filter
%% (cascade of 5 length-3 filters)
H11 = H.^5;
subplot(2,2,1); plot(w, 20*log10(abs(H11)));
xlabel('frequency [rad/sample]');
ylabel('magnitude [dB]');
axis([0,pi,-90,0]); grid;
eval(myreplot);
pause;
%% pole-zero plot
%% In Matlab, it can be done with
%% the single line:
%% zplane(roots([a0,a1,a0]),0);
```

```
w_all = [0:0.05:2*pi];
subplot(2,2,2); plot(exp(i*w_all), '.'); hold on;
zeri = roots([a0, a1, a0]);
plot(real(zeri),imag(zeri), 'o');
plot(0,0, 'x'); hold off;
xlabel('Re');
ylabel('Im');
axis ([-1.2, 1.2, -1.2, 1.2]);
if (plat=='matlab') axis ('square'); end;
eval(myreplot);
pause;
k = [0:10]'; kernelw = exp(-i*k*w);
aa = H11 / kernelw
subplot(2,2,3); plot([0:10],real(aa),'+');
xlabel('samples');
ylabel('h');
grid;
axis;
eval(myreplot);
aa2 = conv([a0 a1 a0],[a0 a1 a0]);
aa3 = conv(aa2,[a0 a1 a0]);
aa4 = conv(aa3,[a0 a1 a0]);
aa5 = conv(aa4,[a0 a1 a0])
%% verify that aa5 = aa:
%%   by composition of convolutions we get
%%   the same length-11 filter
```

In the first couple of lines the script converts the specifications for a length-3 FIR filter. Then, this elementary filter is designed using the technique previously presented in this section. The frequency response H_{11} of the length-11 filter is obtained by exponentiation of the length-3 filter to the fifth power. The magnitude of the frequency response is depicted in fig. 10. We see that the specifications are met. However, the response is not monotonically decreasing. This is due to the fact that the specifications are quite demanding, thus impeding the satisfaction of (21). In fact, the coefficients turn out to be $a_0 = 0.369$ and $a_1 = 0.262$, and the zeros are not real but complex conjugate, as shown in the pole-zero plot of fig. 11. The impulse response of the 10-th order FIR filter is obtained from its frequency response by solving in $[a_0 a_1 \dots a_{10}]$ the matrix

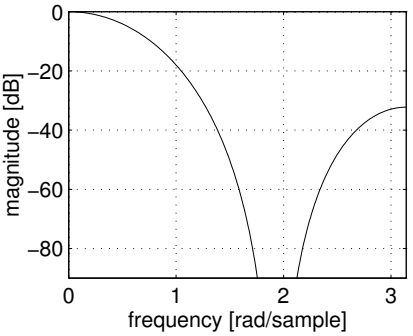

Figure 10: Magnitude of the frequency response of the length-11 filter

equation

$$[a_0 a_1 \dots a_{10}] \begin{bmatrix} 1 \\ e^{-j\omega} \\ \dots \\ e^{-j10\omega} \end{bmatrix} = H_{11}(\omega) \,, \tag{23}$$

which is all contained in the lines

```
k = [0:10]'; kernelw = exp(-i*k*w);
aa = H11 / kernelw;
```

Finally, the ending lines of the script aim at verifying that the same impulse response can be obtained by iterated convolution of the 2-nd order impulse response. The length-11 impulse response is shown in fig. 12.

###

2.1.4 Realizations of FIR Filters

The digital filters, especially FIR filters, are implementable as a sequence of operations "multiply-and-accumulate", often called MAC. In order to run an N-th order FIR filter we need to have, at any instant, the current input sample together with the sequence of the N preceding samples. These N samples constitute the memory of the filter. In practical implementations, it is customary to allocate the memory in contiguous cells of the data memory or, in any case, in locations that can be easily accessed sequentially. At every sampling instant, the state must be updated in such a way that $x(k)$ becomes $x(k-1)$, and this

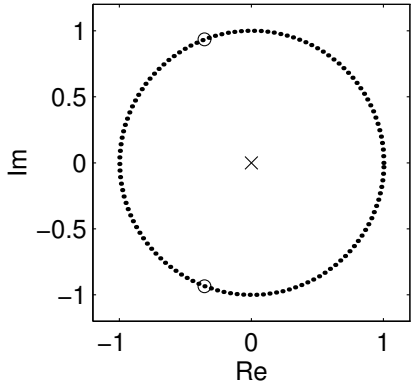

Figure 11: Pole-zero plot for the length-3 FIR filter

seems to imply a shift of N data words in the filter memory. Indeed, instead of moving data, it is convenient to move the indexes that access the data. Consider the scheme depicted in fig. 13, which represents the realization of an FIR filter of order 3. The three memory words are put in an area organized as a circular buffer. The input is written to the word pointed by the index and the three preceding values of the input are read with the three preceding values of the index. At every sample instant, the four indexes are incremented by one, with the trick of beginning from location 0 whenever we exceed the length M of the buffer (this ensures the circularity of the buffer). The counterclockwise arrow indicates the direction taken by the indexes, while the clockwise arrow indicates the movement that should be done by the data if the indexes would stay in a fixed position. In fig. 13 we use small triangles to indicate the multiplications by the filter coefficients. This is a notation commonly used for multiplications within the signal flowgraphs that represent digital filters. As a matter of fact, an FIR filter contains a delay line since it stores N consecutive samples of the input sequence and uses each of them with a delay of N samples at most. The points where the circular buffer is read are called taps and the whole structure is called a tapped delay line.

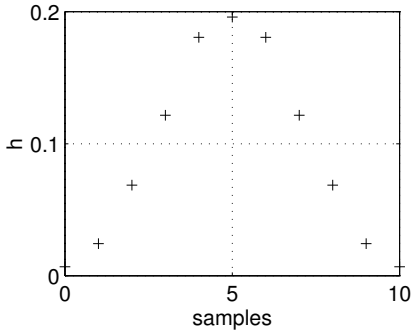

Figure 12: Impulse response of the length-11 FIR filter

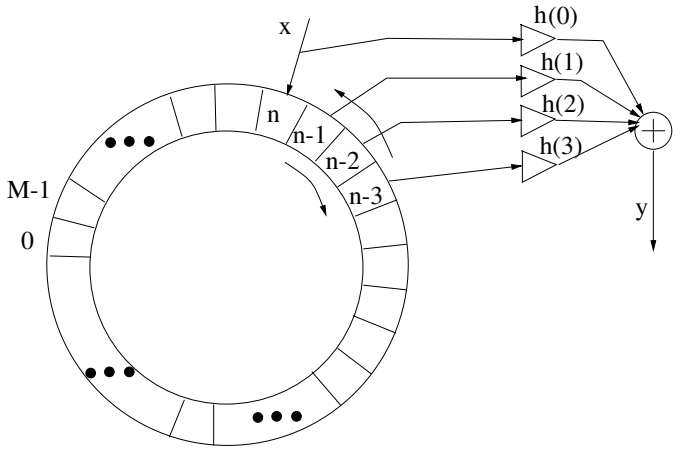

Figure 13: Circular buffer that implements a 3-rd order FIR filter

2.2 IIR Filters

In general, a causal IIR filter is represented by a difference equation where the output signal at a given instant is obtained as a linear combination of samples of the input and output signals at previous time instants. Moreover, an instantaneous dependency of the output on the input is also usually included in the IIR filter. The difference equation that represents an IIR filter is

$$y(n) = -\sum_{m=1}^{N} a_m y(n-m) + \sum_{m=0}^{M} b_m x(n-m) \ . \tag{24}$$

Eq. (24) is also called Auto-Regressive Moving Average (ARMA) representation. While the impulse response of FIR filters has a finite time extension, the impulse response of IIR filters has, in general, an infinite extension. The transfer function is obtained by application of the Z transform to the sequence (24). In virtue of the shift theorem, the Z transform is a mere operatorial substitution of each translation by m samples with a multiplication by z^{-m}. The result is the rational function $H(z)$ that relates the Z transform of the output to the Z transform of the input:

$$Y(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} X(z) = H(z)X(z) \ . \tag{25}$$

The filter order is defined as the degree of the polynomial in z^{-1} that is the denominator of (25).

2.2.1 The Simplest IIR Filter

In this section we analyze the properties of the simplest nontrivial IIR filter that can be conceived: the one-pole filter having coefficients $a_1 = -\frac{1}{2}$ and $b_0 = \frac{1}{2}$:

$$y(n) = \frac{1}{2} y(n-1) + \frac{1}{2} x(n) \ . \tag{26}$$

The transfer function of this filter is

$$H(z) = \frac{1/2}{1 - \frac{1}{2} z^{-1}} \ . \tag{27}$$

If the filter (26) is fed with a unit impulse at instant 0, the response will be:

$$y = 0.5, 0.25, 0.125, 0.0625, \ldots \ . \tag{28}$$

It is clear that the impulse response is nonzero over an infinitely extended support, and every sample is obtained by halving the preceding one. Similarly to what we did for the first-order FIR filter, we analyze the behavior of this filter using a complex sinusoid having magnitude A and initial phase ϕ, i.e. the signal $Ae^{j(\omega_0 n+\phi)}$. Since the system is linear, we do not loose any generality by considering unit-magnitude signals ($A = 1$). Moreover, since the system is time invariant, we do not loose generality by considering signals having the initial phase set to zero ($\phi = 0$). In a linear and time-invariant system, the steady-state response to a complex sinusoidal input is a complex sinusoidal output. To have a confirmation of that, we can consider the reversed form of (26)

$$x(n) = 2y(n) - y(n-1) \; , \tag{29}$$

and replace the output $y(n)$ with a complex sinusoid, thus obtaining

$$x(n) = 2e^{j\omega_0 n} - e^{j\omega_0(n-1)} = (2 - e^{-j\omega_0})y(n) \; . \tag{30}$$

Eq. (30) shows that a sinusoidal output gives a sinusoidal input, and vice versa. The input sinusoid gets rescaled in magnitude and shifted in phase. Namely, the output y is a copy of the input multiplied by the complex quantity $\frac{1}{2-e^{-j\omega_0}}$, which is the value taken by the transfer function (27) at the point $z = e^{j\omega_0}$. The frequency response is

$$H(\omega) = \frac{1/2}{1 - \frac{1}{2}e^{-j\omega}} \; , \tag{31}$$

and there are no simple formulas to express its magnitude and phase, so that we have to resort to the graphical representation, depicted in fig. 14. This simple filter still has a lowpass shape. As compared to the first-order FIR filter, the one-pole filter gives a steeper magnitude response curve. The fact that, for a given filter order, the IIR filters give a steeper (or, in general, a more complex) frequency response is a general property that can be seen as an advantage in preferring IIR over FIR filters. The other side of the coin is that IIR filters can not have a perfectly-linear phase. Furthermore, IIR filters can produce numerical artifacts, especially in fixed-point implementations.

The one-pole filter can also be analyzed by watching its pole-zero distribution on the complex plane. To this end, we rewrite the transfer function as a ratio of polynomials in z and give a name to the root of the denominator: $p_0 = \frac{1}{2}$. The transfer function has the form

$$H(z) = \frac{1}{2}\frac{z}{z - \frac{1}{2}} = \frac{1}{2}\frac{z}{z - p_0}. \tag{32}$$

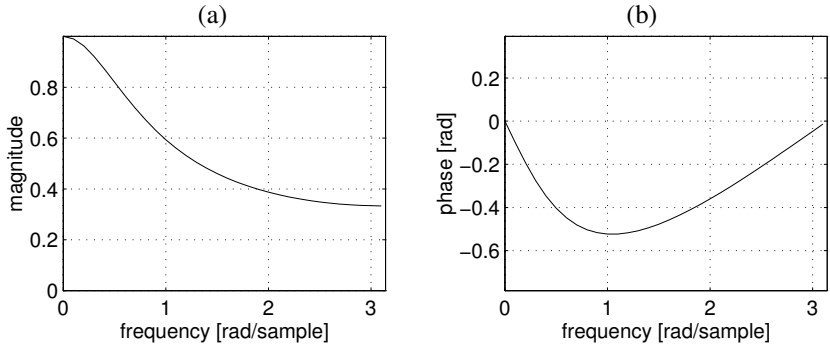

Figure 14: Frequency response (magnitude (a) and phase (b)) of a one-pole IIR filter

We can apply the graphic method presented in sec. 2.1.1 to have a qualitative idea of the magnitude and phase responses. In order to do that, we consider the point $e^{j\omega}$ on the unit circle as the head of the vectors that connect it to the pole p_0 and to the zero in the origin. Fig. 15 is illustrative of the procedure. While we move along the unit circumference from dc to the Nyquist frequency, we go progressively away from the pole, and this is reflected by the monotonically decreasing shape of the magnitude response.

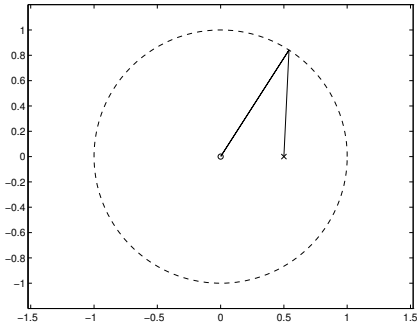

Figure 15: Single pole ($\times$) and zero in the origin ($\circ$)

To have a complete picture of the filter behavior we need to analyze the transient response to the causal complex exponential. The Z transform of the

input has the well-known form

$$X(z) = \frac{1}{1 - e^{j\omega_0} z^{-1}} \ . \tag{33}$$

A multiplication of $X(z)$ by $H(z)$ in the Z domain gives

$$\begin{aligned} Y(z) &= H(z)X(z) = \frac{1}{2} \frac{1}{1 - \frac{1}{2} z^{-1}} \frac{1}{1 - e^{j\omega_0} z^{-1}} \\ &= \frac{1/2}{1 - 2e^{j\omega_0}} \frac{1}{1 - \frac{1}{2} z^{-1}} + \frac{1/2}{1 - 1/2e^{-j\omega_0}} \frac{1}{1 - e^{j\omega_0} z^{-1}} \ , \end{aligned} \tag{34}$$

where we have done a partial fraction expansion of $Y(z)$. The second addendum of the last member of (34) represents the steady-state response, and it is the product of the Z transform of the causal complex exponential sequence by the filter frequency response evaluated at the same frequency of the input signal. The first addendum of the last member of (34) represents the transient response and it can be represented as a causal exponential sequence:

$$y_t(n) = A {p_0}^n \ , \tag{35}$$

where $A = \frac{1/2}{1 - 2e^{j\omega_0}}$. Since $|p_0| < 1$ (i.e., the pole is within the unit circle), the transient response is doomed to die out for increasing values of n. In general, for causal systems, the stability condition (29) of chapter 1 is shown to be equivalent to having all the poles within the unit circle. If the condition is not satisfied, even if the steady-state response is bounded, the transient will diverge. In terms of Z transform, a system is stable if the region of convergence is a geometric ring containing the unit circumference; the system is causal if such ring extends to infinity out of the circle, and it is anticausal if it extends down to the origin.

It is useful to evaluate the time needed to exhaust the initial transient. We define the time constant τ_n (in samples) of the filter as the time taken by the exponential sequence ${p_0}^n$ to reduce its amplitude to 1% of the initial value. We have

$${p_0}^{\tau_n} = 0.01 \ , \tag{36}$$

and, therefore,

$$\tau_n = \frac{\ln 0.01}{\ln p_0} \ , \tag{37}$$

where the logarithm can be evaluated in any base. In our example, where $p_0 = 1/2$, we obtain $\tau_n \approx 6.64$ samples. The time constant in seconds τ is obtained

by multiplication of τ_n by the sampling rate. This way of evaluating the time constant corresponds to evaluating the time needed to attenuate the transient response by 40dB. When we refer to systems for artificial reverberation such lower threshold of attenuation is moved to 60dB, thus corresponding to 0.1% of the initial amplitude of the impulse response.

In the case of higher-order IIR filters, we can always do a partial fraction expansion of the response to a causal exponential sequence, in a way similar to what has been done in (34), where each addendum but the last one corresponds to a single complex pole of the transfer function. The transient response of these systems is, therefore, the superposition of causal complex exponentials, each corresponding to a complex pole of the transfer function. If the goal is to estimate the duration of the transient response, the pole that is closest to the unit circumference is the dominant pole, since its time constant is the longest. It is customary to define the time constant of the whole system as the constant associated with the dominant pole.

2.2.2 Higher-Order IIR Filters

The two-pole IIR filter is a very important component of any sound processing environment. Such filter, which is capable of selecting the frequency components in a narrow range, can find practical applications as an elementary resonator.

Instead of starting from the transfer function or from the difference equation, in this case we begin by positioning the two poles in the complex plane at the point

$$p_0 = re^{j\omega_0} \tag{38}$$

and at its conjugate point $p_0{}^* = re^{-j\omega_0}$. In fact, if p_0 is not real, the two poles must be complex conjugate if we want to have a real-coefficient transfer function. In order to make sure that the filter is stable, we impose $|r| < 1$. The transfer function of the second-order filter can be written as

$$\begin{aligned} H(z) &= \frac{G}{(1 - re^{j\omega_0}z^{-1})(1 - re^{-j\omega_0}z^{-1})} \\ &= \frac{G}{1 - r(e^{j\omega_0} + e^{-j\omega_0})z^{-1} + r^2z^{-2}} = \frac{G}{1 - 2\,r\,cos\omega_0 z^{-1} + r^2z^{-2}} \\ &= \frac{G}{1 + a_1z^{-1} + a_2z^{-2}} \end{aligned} \tag{39}$$

where G is a parameter that allows us to control the total gain of the filter.

As usual, we obtain the frequency response by substitution of z with $e^{j\omega}$ in (31):

$$H(\omega) = \frac{G}{1 - 2\,r\,cos\omega_0 e^{-j\omega} + r^2 e^{-2j\omega}} \; . \tag{40}$$

If the input is a complex sinusoid at the (resonance) frequency ω_0, the output is, from the first of (39):

$$\begin{aligned} H(\omega_0) &= \frac{G}{(1-r)(1-re^{-2j\omega_0})} = \\ &= \frac{G}{(1-r)(1-r\cos 2\omega_0 + j\,r\sin 2\omega_0)} \; . \end{aligned} \tag{41}$$

In order to have a unit-magnitude response at the frequency ω_0 we have to impose

$$|H(\omega_0)| = 1 \tag{42}$$

and, therefore,

$$G = (1-r)\sqrt{1 - 2r\cos 2\omega_0 + r^2} \; . \tag{43}$$

The frequency response of this normalized filter is reported in fig. 16 for $r = 0.95$ and $\omega_0 = \pi/6$. It is interesting to notice the large step experienced by the phase response around the resonance frequency. This step approaches π as the poles get closer to the unit circumference.

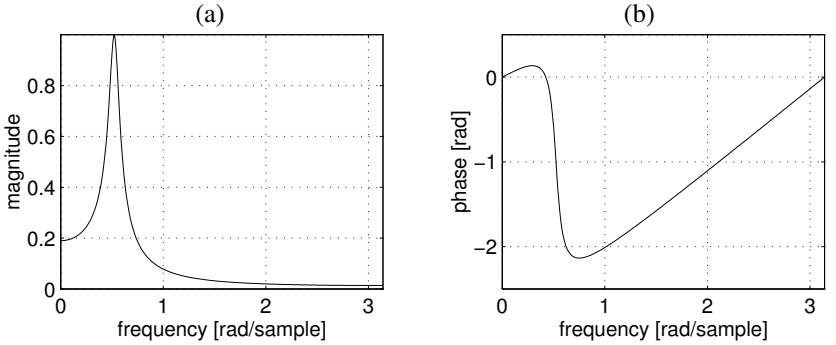

Figure 16: Frequency response (magnitude (a) and phase (b)) of a two-pole IIR filter

It is useful to draw the pole-zero diagram in order to gain intuition about the frequency response. The magnitude of the frequency response is found by

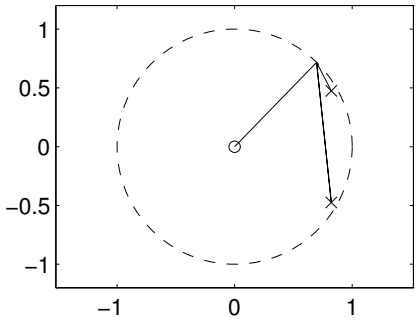

Figure 17: Couple of poles on the complex plane

taking the ratio of the product of the magnitudes of the vectors that go from the zeros to the unit circumference with the product of the magnitudes of the vectors that go from the poles to the unit circumference. The phase response is found by taking the difference of the sum of the angles of the vectors starting from the zeros with the sum of the angles of the vectors starting from the poles. If we move along the unit circumference from dc to the Nyquist rate, we see that, as we approach the pole, the magnitude of the frequency response increases, and it decreases as we move away from the pole. Reasoning on the complex plane it is also easier to figure out why there is a step in the phase response and why the width of this step converges to π as we move the pole toward the unit circumference. In the computation of the frequency response it is clear that, in the neighborhood of a pole close to the unit circumference, the vector that comes from that pole is dominant over the others. This means that, accepting some approximation, we can neglect the longer vectors and consider only the shortest vector while evaluating the frequency response in that region. This approximation is useful to calculate the bandwidth $\Delta\omega$ of the resonant filter, which is defined as the difference between the two frequencies corresponding to a magnitude attenuation by $3dB$, i.e., a ratio $1/\sqrt{2}$. Under the simplifying assumption that only the local pole is exerting some influence in the neighboring area, we can use the geometric construction of fig. 18 in order to find an expression for the bandwidth [67]. The segment $\overline{P_0A}$ is $\sqrt{2}$ times larger than the segment $\overline{P_0P}$. Therefore, the triangle formed by the points P_0AP has two, orthogonal, equal edges and $AB = 2P_0P = 2(1 - r)$. If AB is small enough, its length can be approximated with that of the arc subtended by it, which is the bandwidth that we are looking for. Summarizing, for poles that

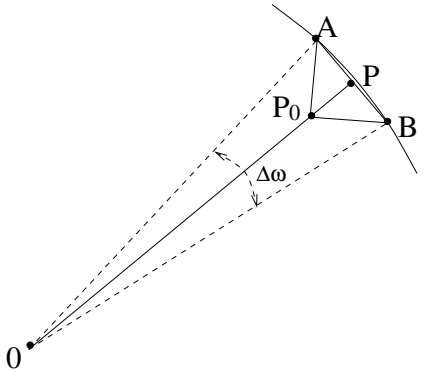

Figure 18: Graphic construction of the bandwidth. P_0 is the pole. $\overline{P_0P} \approx 1-r$.

are close to the unit circumference, the bandwidth is given by

$$\Delta\omega = 2(1-r)\ . \tag{44}$$

The formula (44) can be used during a filter design stage in order to guide the pole placement on the complex plane.

The transfer function (39) can be expanded in partial fractions as

$$\begin{aligned} H(z) &= \frac{G}{(1-re^{j\omega_0}z^{-1})(1-re^{-j\omega_0}z^{-1})} \\ &= \frac{G/(1-e^{-j2\omega_0})}{1-re^{j\omega_0}z^{-1}} - \frac{Ge^{-j2\omega_0}/(1-e^{-j2\omega_0})}{1-re^{-j\omega_0}z^{-1}}\ , \end{aligned} \tag{45}$$

and each addendum is the Z transform of a causal complex exponential sequence. By manipulating the two sequences algebraically and expressing the sine function as the difference of complex exponentials we can obtain the analytic expression of the impulse response[8]

$$h(n) = \frac{Gr^n}{\sin\omega_0}\sin\left(\omega_0 n + \omega_0\right)\ . \tag{46}$$

The impulse response is depicted in fig. 19, which shows that a resonant filter can be interpreted in the time domain as a damped oscillator with a characteristic frequency that corresponds to the phase of the poles in the complex plane.

[8]The reader is invited to work out the expression (46).

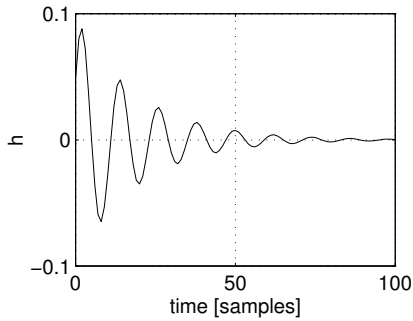

Figure 19: Impulse response of a second-order resonant filter

As we have anticipated in sec. 2.2.1, the time constant is determined by evaluating the distance of one of the poles from the unit circumference. In the specific case that we are examining, such a time constant is

$$\tau_n = \frac{\ln 0.01}{\ln r} = \frac{\ln 0.01}{\ln 0.95} \approx 90 \text{ samples} , \tag{47}$$

and we can verify from fig. 19 that this value makes sense.

Example 3. With the example that follows we face the problem of doing a practical implementation of a filter. The platform that we adopt is the CSound language (see appendix B.2) and our prototypical implementation is the second-order all-pole IIR filter. This simple example can be extended to higher-order filters.

We design an "orchestra" of two instruments: an excitation instrument and a filtering block. The excitation block generates white noise. The filtering block extracts from the noise the components in a band around a center frequency, passed as a parameter, that corresponds to the phase of the pole[9]. Another parameter is the decay time of the response of the resonant filter, which is related to the resonance bandwidth. The Csound orchestra that implements our two blocks is:

```
; res.orc: by Francesco Scagliola and Davide Rocchesso
        sr=44100
```

[9]Indeed, the central frequency of the passing frequency band is not coincident with the phase of the complex pole, since the conjugate pole can exert some influence and slightly modify the frequency response in the neighborood of the other pole. However, for our purposes it is not dangerous to mix the two concepts, provided that the resulting spectrum corresponds to our needs.

```
        kr=44100
        ksmps=1
        nchnls=1
ga1     init 0
gamp    init 30000

        instr 1

        ; white noise generator
a1      rand gamp
ga1     =    a1 ; sound to be passed to the filter

        endin

        instr 2

; p4    central frequency
; p5    decay time

ipi     =  3.141592654
ithres  =  0.01
        ; the duration of the frequency response
 ; is measured in seconds until the response
 ; goes below the threshold 20*log10(ithres)
        ; [-40 dB]
iw0     = 2*ipi*p4/sr
        ; frequency correspondent to the pole phase
ir      = exp((1/(sr*p5))*log(ithres))
        ; radius of the pole
ia1     = -2*ir*cos(iw0)
        ; coefficient a1 of the filter denominator
ia2     = ir*ir
        ; coeff. a2 of the filter denominator
ig      = (1-ir)*sqrt(1-2*ir*cos(2*iw0)+ir*ir)* \
          10*sqrt(p5)
        ; coefficient to have unit gain at the
        ; center of the band

izero   = 0
```

```
as1     init izero ; initialize the filter status
as2     init izero

afilt   = -ia1*as1-ia2*as2+ig*ga1
        ; difference equation

out     afilt

as2     = as1   ; filter status update
as1     = afilt

        endin
```

The orchestra can be experimented with the score

```
;instr. time    durat.  freq.   decay
i1      0       30.0
i2      0       5       700     0.1
i2      5       5       700     1.0
i2      10      5       1700    0.2
i2      15      5       2900    2.0
i2      20      5       700     1.0
i2      20      5       1700    1.5
i2      20      5       2900    2.0
```

The sounds resulting from the score performance are represented in the sonogram of fig. 20, where larger magnitudes are represented by darker points. In the filtering instrument, the filter coefficients are computed according to the formulas (47) and (39), starting from the given decay time and central frequency. Moreover, the signal is rescaled by a gain such that the magnitude of the frequency response is one at the central frequency. Empirically, we have found that, in order to keep some homogeneity in the output energy level even for very narrow frequency responses, it is useful to insert a further factor equal to ten times the square root of the decay time. Another observation concerns the difference equation. This equation uses two state variables `as1` and `as2`, used to store the previous values of the output. The state variables are updated in the final two lines of the instrument.

It is interesting to reduce the control rate in the orchestra, for instance by a factor ten. The resulting sounds will have fundamental frequencies lowered by the same factor and the spectrum will be repeated at multiples of `sr/10`.

This kind of artifacts is often found when writing explicit filtering structures in CSound and using a sample rate different from the control rate. The reason for such a strange behavior is found in the special block processing used by the CSound interpreter, which uses `sr/kr` variables for each signal variable indicated in the orchestra, and updates all these variables in the same cycle. This means that, as a matter of fact, we get `sr/kr` filters, each working at a reduced sample rate on a signal undersampled by a factor `sr/kr`. The samples of the partial results are then interleaved to give the signal at the sampling rate `sr`. The output of each of the undersampled filters is subject to an upsampling that produces the `sr/kr` periodic replicas of the spectrum.

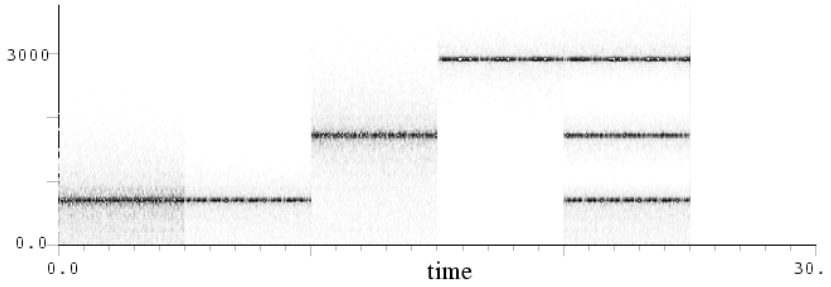

Figure 20: Sonogram of a musical phrase produced by filtering white noise

###

Positioning the zeros

We have seen how the poles can be positioned within the unit circle in order to give resonances at the desired frequency and with the desired bandwidth. The ratio between the central frequency and the width of a band is often called quality factor and indicated with the symbol Q.

In many cases, it is necessary to design a filter having a flat frequency response (in magnitude) except for a narrow zone around a frequency ω_0 where it amplifies or attenuates. The resonant filter that we have just introduced can be modified for this purpose by introducing a couple of zeros positioned near the poles. In particular, the numerator of the transfer function will be the polynomial in z^{-1} having roots at $z_0 = r_0 e^{j\omega_0}$ and at ${z_0}^* = r_0 e^{-j\omega_0}$. By means of a qualitative analysis of the pole-zero diagram we can realize that, if $r_0 < r$ we

have a boost of the frequency response, and if $r_0 > r$ we have an attenuation (a notch) of the response around ω_0. The reader is invited to do this qualitative analysis on her own and to write the Octave/Matlab script that produces fig. 21, which is obtained using the values $r_0 = 0.9$ and $r_0 = 1.0$. We notice that the phase jumps down by 2π radians when we cross a zero laying on the unit circumference.

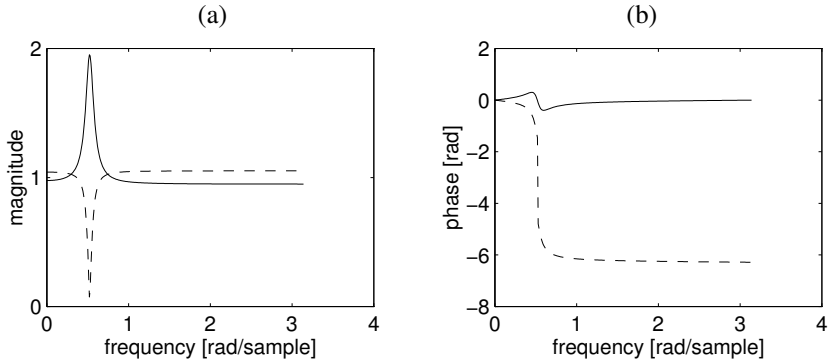

Figure 21: Frequency response (magnitude and phase) of an IIR filter with two poles ($r = 0.95$) and two zeros. The notch filter (dashed line) has the zeros with magnitude 1.0. The boost filter (solid line) has the zeros with magnitude 0.9.

2.2.3 Allpass Filters

Imagine that we are designing a filter by positioning its poles within the unit circle in the complex plane. For each complex pole p_i, let us introduce a zero $z_i = 1/{p_i}^*$ in the transfer function. In other words, we form the pole-zero couple

$$H_i(z) = \frac{z^{-1} - {p_i}^*}{1 - p_i z^{-1}} \ , \tag{48}$$

which places the pole and the zero on reciprocal points about the unit circumference and along tha same radius that links them to the origin. Moving along the circumference we can realize that the vectors drawn from the pole and the zero have lengths that keep a constant ratio. A more accurate analysis can be done using the frequency response of this pole-zero couple, which is written as

$$H_i(\omega) = \frac{e^{-j\omega} - {p_i}^*}{1 - p_i e^{-j\omega}} = e^{-j\omega} \frac{1 - {p_i}^* e^{j\omega}}{1 - p_i e^{-j\omega}} \ . \tag{49}$$

It is clear that numerator and denominator of the fraction in the last member of (49) are complex conjugate one to each other, thus meaning that the rational function has unit magnitude at any frequency. Therefore, the couple (49) is the fundamental block for the construction of an allpass filter, whose frequency response is obtained by multiplication of blocks such as (49).

The allpass filters are systems that leave all frequency component magnitudes unaltered. Stationary sinusoidal input signals can only be subject to phase delays, with no modification in magnitude. The phase response and phase delay of the fundamental pole-zero couple are depicted in fig. 22 for values of pole set to $p_1 = 0.9$ and $p_1 = -0.9$. A second-order allpass fil-

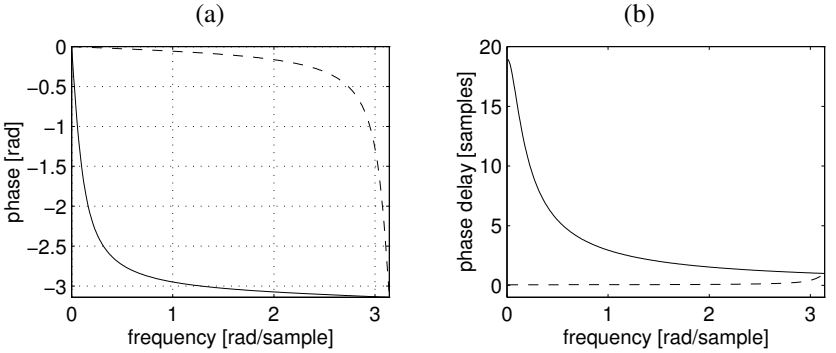

Figure 22: Phase of the frequency response (a) and phase delay (b) for a first-order allpass filter. Pole in $p_1 = 0.9$ (solid line) and pole in $p_1 = -0.9$ (dashed line)

ter with real coefficients is obtained by multiplication of two allpass pole-zero couples, where the poles are the conjugate of each other. Fig. 23 shows the phase response and the phase delay of a second order allpass filter with poles in $p_1 = 0.9 + i0.2$ and $p_2 = 0.9 - i0.2$ (solid line) and in $p_1 = -0.9 + i0.2$ and $p_2 = -0.9 - i0.2$ (dashed line). It can be shown that the phase response of any allpass filter is always negative and monotonically decreasing [65]. The group and phase delays are always functions that take positive values. This fact allows us to think about allpass filters as media where signals propagate with a frequency-dependent delay, without being subject to any absorption or

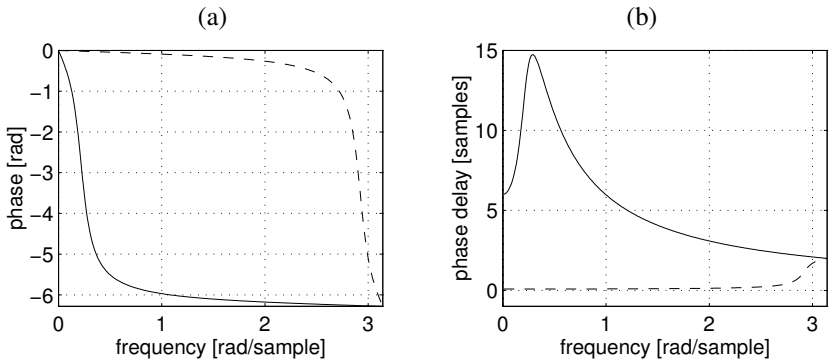

Figure 23: Phase of the frequency response (a) and phase delay (b) for a second-order allpass filter. Poles in $p_{1,2} = 0.9 \pm i0.2$ (solid line) and $p_{1,2} = -0.9 \pm i0.2$ (dashed line)

amplification.

The reader might think that the allpass filters are like open doors for audio signals, since the phase shifts are barely distinguishable by the human hearing system. Actually, this is true only for stationary signals, i.e., signals formed by stable sinusoidal components. Real-world sounds are made of transients at least as much as they are made of stationary components, and the transient response of allpass filters can be characterized according to what we showed in sec. 2.2. During transients, the phase response plays an important role for perception, and in this sense the allpass filters can modify the sound signals appreciably. For instance, very-high-order allpass filters are used to construct artificial reverberators. These filters usually have a long time constant, so that the effects of their phase response are mainly perceived in the time domain in the form of a reverberation tail.

The importance of allpass filters becomes readily evident when they are inserted into complex computational structures, typically to construct filters whose properties should be easy to control. We will see an example of this use of allpass filters in sec. 2.3.

2.2.4 Realizations of IIR Filters

So far, we have studied the IIR filters by analysis of transfer functions or impulse responses. In this section we want to face the problem of implementing

these filters as computational structures that can be directly coded using sound processing languages or real-time sound processing environments.

Consider a second-order filter with two poles and two zeros, which is represented by the transfer function (25) with $N = M = 2$. This can be realized by the signal flowgraph of fig. 24, where the nodes having converging edges are considered as points of addition, and the nodes having diverging edges are considered as branching points. Such a realization is called Direct Form I.

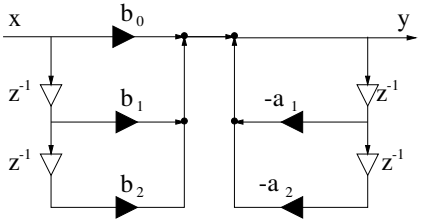

Figure 24: Second-order filter, Direct Form I

Signal flowgraphs can be manipulated in several ways, thus leading to alternative realizations having different numerical properties and, possibly, more computationally efficient. For instance, if we want to implement a filter as a cascade of second-order cells such as that of fig. 24, we can share, between two contiguous cells, the unit delays that are on the output stage of the first cell, with the unit delays that are on the input stage of the second cell, thus saving a number of memory accesses.

We are going to show some other kind of manipulation of signal flowgraphs, in the special case of the realization of the second-order allpass filter, which has the property

$$b_i = a_{2-i}, \quad i = 0, 1, 2 \; . \tag{50}$$

A first transformation comes from the observation that the structure of fig. 24 is formed by the cascade of two blocks, each being linear and time invariant. Therefore, the two blocks can be commuted without altering the input-output behavior. Moreover, from the block exchange we get a flowgraph with two side-to-side stages of pure delays, and these stages can be combined in one only. The realization of these transformations is shown in fig. 25 and it is called Direct Form II.

Another transformation that can be done on a signal flowgraph without altering its input-output behavior is the transposition [65]. The transposition of

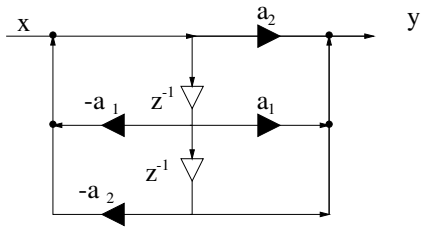

Figure 25: Second-order allpass filter, Direct Form II

a signal flowgraph is done with the following operations:

- Inversion of the direction of all the edges
- Transformation of the nodes of addition into branching nodes, and vice versa
- Exchange of the roles of the input and output edges

The transposition of a realization in Direct Form II leads to the Transposed Form II, which is shown in fig. 26. Similarly, the Transposed Form I is obtained by transposition of the Direct Form I.

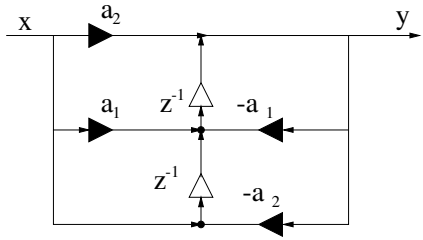

Figure 26: Second-order allpass filter, Transposed Form II

By direct manipulation of the graph, we can also take advantage of the properties of special filters. For instance, in an allpass filter, the coefficients of the numerator are the same of the denominator, in inverted order (see (50)). With simple transformations of the graph of the Direct Form II it is possible to obtain the realization of fig. 27, which is interesting because it only has two multiplies. In fact, the multiplications by -1 can be avoided by replacing two additions with subtractions.

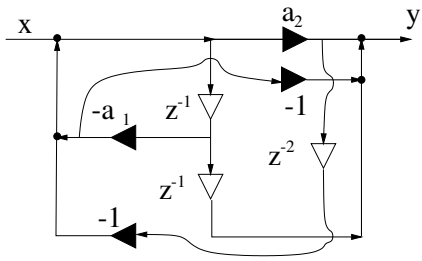

Figure 27: Second-order allpass filter, realization with two multipliers and four state variables

A special structure that plays a very important role in signal processing is the lattice structure, which can be used to implement FIR and IIR filters [65]. In particular, the IIR lattice filters are interesting because they have physical analogues that can be considered as physical sound processing systems. The lattice structure can be defined in a recursive fashion as indicated in fig. 28, where $H_{a\,M-1}$ is an order $M-1$ allpass filter, k_M is called reflection coefficient and it is a real number not exceeding one. Between the signals x and y there is an

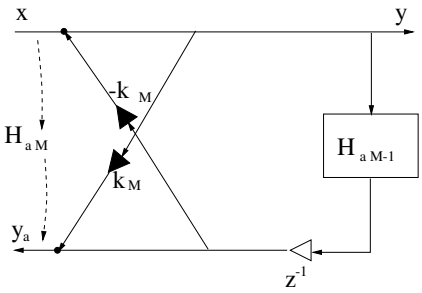

Figure 28: Lattice filter

all-pole transfer function $1/A(z)$, while between the points x and y_a there is an allpass transfer function $H_{a\,M}(z)$ having the same denominator $A(z)$. More precisely, it can be shown that, if $H_{a\,M-1}$ is an allpass stable transfer function and $|k_M| < 1$, then $H_{a\,M}$ is an allpass stable transfer function. Proceeding with the recursion, the allpass filter $H_{a\,M-1}$ can be realized as a lattice structure, and so on. The recursion termination is obtained by replacing $H_{a\,1}$ with a short circuit. The lattice section having coefficient k_M can be interpreted as the

junction between two cylindrical lossless tubes, where k_M is the ratio between the two cross-sectional areas. This number is also the scaling factor that an incoming wave is subject to when it hits the junction, so that the name reflection coefficient is justified. To have a physical understanding of lattice filters, think of modeling the human vocal tract. The lattice realization of the transfer function that relates the signals produced by the vocal folds to the pressure waves in the mouth can be interpreted as a piecewise cylindrical approximation of the vocal tract. In this book, we do not show how to derive the reflection coefficients from a given transfer function [65]. We just give the result that, for a second-order filter, a denominator such as $A(z) = 1 + a_1 z^{-1} + a_2 z^{-2}$ gives the reflection coefficients[10]

$$\begin{aligned} k_1 &= a_1/(1+a_2) \\ k_2 &= a_2 \ . \end{aligned} \tag{51}$$

[10] Verify that the filter is stable if and only if $|k_1| < 1$ and $|k_2| < 1$.

2.3 Complementary filters and filterbanks

In sec. 2.2.4 we have presented several different realizations of allpass filters because they find many applications in signal processing [76]. In particular, a couple of allpass filters is often combined in a parallel structure in such a way that the overall response is not allpass. If H_{a1} and H_{a2} are two different allpass filters, their parallel connection, having transfer function $H_l(z) = H_{a1}(z) + H_{a2}(z)$ is not allpass. To figure this out, just think about frequencies where the two phase responses are equal. At these points the signal will be doubled at the output of $H(z)$. On the other hand, at points where the phase response are different by π (i.e., they are in phase opposition), the outputs of the two branches cancel out at the output. In order to design a lowpass filter it is sufficient to connect in parallel two allpass filters having a phase response similar to that of fig. 29.The same parallel connection, with a subtraction instead of the addition at the output, gives rise to a highpass filter $H_h(z)$, and it is possible to show that the highpass and the lowpass transfer functions are complementary, in the sense that $|H_l(\omega)|^2 + |H_h(\omega)|^2$ is constant in frequency. Therefore, we have the compact realization of a crossover filter, as depicted in

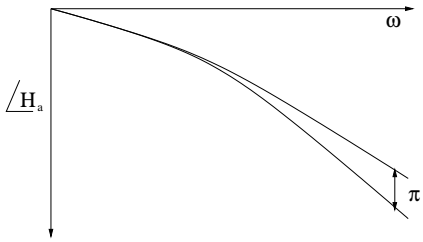

Figure 29: Phase responses of two allpass filters that, if connected in parallel, give a lowpass filter

fig. 30, which is a device with one input and two outputs that conveys the low frequencies to one outlet, and the high frequencies to the other outlet. Devices such as this are found not only in loudspeakers, but also in musical instrument models. For instance, the bell of woodwinds transmits to the air the high frequencies and reflects the low frequencies back to the bore.

The idea of connecting two allpass filters in parallel can be applied to the realization of resonant complementary filters. In particular, it is interesting to be able to tune the bandwidth and the center frequency independently. To construct such a filter, one of the two allpass filters is replaced by the identity (i.e.,

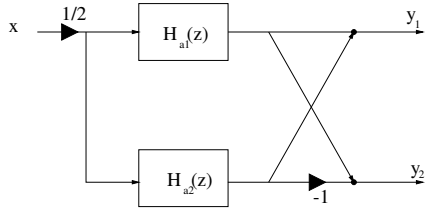

Figure 30: Crossover implemented as a parallel of allpass filters and a lattice junction

a short circuit) while the other one is a second order allpass filter (see fig. 31). Recall that, close to the frequency ω_0 that corresponds to the pole of the filter, the phase response takes values that are very close to $-\pi$ (see fig. 23). Therefore, the frequency ω_0 corresponds to a minimum in the overall frequency response. In other words, it is the notch frequency. The closer is the pole to the unit circumference, the narrower is the notch. The lattice implementation of this allpass filter allows to tune the notch position and width independently, since the two reflection coefficients have the form [76]

$$\begin{aligned} k_1 &= -\cos\omega_0 \\ k_2 &= \frac{1-\tan B/2}{1+\tan B/2}\,, \end{aligned} \tag{52}$$

where B is the bandwidth for 3dB of attenuation.

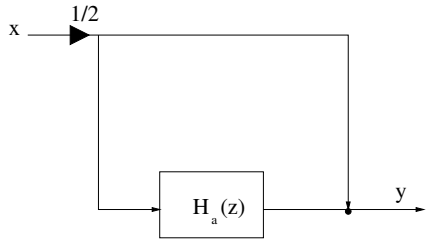

Figure 31: Notch filter implemented by means of a second-order allpass filter

A structure that allows to convert a notch into a boost with a continuous control is obtained by a weighted combination of the complementary outputs and it is shown in fig 32. For values of k such that $0 < k < 1$ the filter is a notch, while for $k > 1$ the filter is a boost.

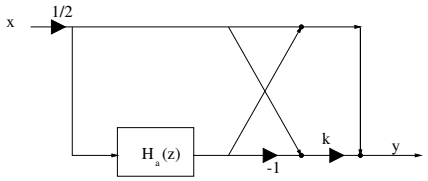

Figure 32: Notch/boost filter implemented by means of a second-order allpass filter and a lattice section

Filters such as those of figures 31 and 32, whose properties can be controlled by a few parameters decoupled with each other, are called parametric filters. For thorough surveys on structures for parametric filtering, with analyses of numerical properties in fixed-point implmentations, we refer the reader to a book by Zölzer [109] and an article by Dattorro [29].

2.4 Frequency warping

Section (1.5.2) has shown how the bilinear transformation distorts the frequency axis while maintaining the "shape" of the frequency response. Such transformation is a so-called conformal transformation [62] of the complex plane onto itself. In this section we are interested in conformal transformations that map the unit circumference (instead of the imaginary axis) onto itself, in such a way that, if applied to a discrete-time filter, they give a new discrete-time filter having the same stability properties.

Indeed, the simplest non-trivial transformation of this kind is a bilinear transformation

$$z^{-1} = \frac{a + \theta^{-1}}{1 + a\theta^{-1}} \; . \tag{53}$$

The transformation (53) is allpass and, therefore, it maps the unit circumference onto itself. Moreover, if the transformation (53) is applied to a discrete-time filter described by a transfer function in z, it preserves the filter order in the variable θ.

The reason for using conformal maps in digital filter design is that it might be easier to design a filter using a warped frequency axis. For instance, to design a presence filter it is convenient to start from a second-order resonant filter prototype having center frequency at $\pi/2$ and tunable bandwidth and boost. Then, it is possible to compute the coefficient of the conformal trans-

formation (53) in such a way that the resonant peak gets moved to the desired position [62]. Conformal transformations of order higher than the first are often used to design multiband filters starting from the design of a lowpass filter, or to satisfy demanding specifications on the slope of the transition band that connects the pass band from the attenuated band.

When designing digital filters to be used in models of acoustic systems, the transformation (53) can be useful, especially if it is specialized in order to optimize some psychoacoustic-based quality measure. Namely, the warping of the frequency axis can be tuned in such a way that it resembles the frequency distribution of critical bands in the basilar membrane of the ear [99]. Similarly to what we saw in section 1.5.2 for the bilinear transformation, it can be shown that a first-order conformal map is determined by setting the correspondence in three points, two of them being $\omega = 0$ and $\omega = \pi$. The mapping of the third point is determined by the coefficient a to be used in (53). Surprisingly enough, a simple first-order transformation is capable to follow the distribution of critical bands quite accurately. Smith and Abel [99], using a technique that minimizes the squared equation error, have estimated the value that has to be assigned to a for sampling frequencies ranging from 1Hz to 50KHz, in order to have a ear-based frequency distribution. An approximate expression to calculate such coefficient is

$$a(F_s) \simeq 1.0211 \left[\frac{2}{\pi} \arctan\left(76 \cdot 10^{-6} F_s\right)\right]^{1/2} - 0.19877 \,. \qquad (54)$$

As an exercise, the reader can set a value of the sampling rate F_s, and compute the value of a by means of (54). Then the curve that maps the frequencies in the θ plane to the frequencies in the z plane can be drawn and compared to the curve obtained by uniform distribution of the center frequencies of the Bark scale[11] [99, 111] that are below the Nyquist rate.

A psychoacoustics-driven frequency warping is also useful to design digital filters in such a way that the approximation error gets distributed on the frequency axis in a way that is most tolerable by our ears. The procedure consists in transforming the desired frequency response according to (53), and designing a digital filter that approximates it using some filter design method [65]. Then the inverse conformal mapping (unwarping) is applied on the resulting

[11] The Bark scale is based on measurements on critical bands, published by Zwicker in 1961. The center frequencies (in Hz) of the rectangular filters, equivalent in power to the critical bands, are: 50, 150, 250, 350, 450, 570, 700, 840, 1000, 1170, 1370, 1600, 1850, 2150, 2500, 2900, 3400, 4000, 4800, 5800, 7000, 8500, 10500, 13500, 20500, 27000.

digital filter. Some filter design techniques, beyond giving a better approximation in a psychoacoustic sense, take advantage of the expansion of low frequencies induced by the warping map, because low-frequency sharp transitions get smoother and the design algorithms become less sensitive to numerical errors.

Chapter 3

Delays and Effects

Most acoustic systems have some component where waves can propagate, such as a membrane, a string, or the air in an enclosure. If propagation in these media is ideal, i.e., free of losses, dispersion, and nonlinearities, it can be simulated by delay lines.

A delay line is a linear time-invariant, single-input single-output system, whose output signal is a copy of the input signal delayed by τ seconds. In continuous time, the frequency response of such system is

$$H_{D_s}(j\Omega) = e^{-j\Omega\tau} \ . \tag{1}$$

Equation (1) tells us that the magnitude response is unitary, and that the phase is linear with slope τ.

3.1 The Circular Buffer

A discrete-time realization of the system (1) is given by a system that implements the transfer function

$$H_D(z) = z^{-\tau F_s} \triangleq z^{-m} \ , \tag{2}$$

where m is the number of samples of delay. When the delay τ is an integral multiple of the sampling quantum, m is an integer number and it is straightforward to implement the system (2) by means of a memory buffer. In fact, an m-samples delay line can be implemented by means of a circular buffer, that is

a set of M contiguous memory cells accessed by a write pointer IN and a read pointer OUT, such that

$$\text{IN} = (\text{OUT} + m)\%M \ , \tag{3}$$

where the symbol $\%$ is used for the quotient modulo M. At each sampling instant, the input is written in the location pointed by IN, the output is taken from the location pointed by OUT, and the two pointers are updated with

$$\begin{array}{lcl} \text{IN} & = & (\text{IN} + 1)\%M \\ \text{OUT} & = & (\text{OUT} + 1)\%M \end{array} \ . \tag{4}$$

In words, the pointers are incremented respecting the circularity of the buffer.

In some architectures dedicated to sound processing, memory organization is optimized for wavetable synthesis, where a stored waveform is read with variable increments of the reading pointer. In these architectures, a quantity of 2^r memory locations is available, and from these $M = 2^s$ locations (with $s < r$) are uniformly chosen among the 2^r available cells. In this case the locations of the circular buffer are not contiguous, and the update of the pointers is done with the operations

$$\begin{array}{lcl} \text{IN} & = & (\text{IN} + 2^{r-s})\%2^r \\ \text{OUT} & = & (\text{OUT} + 2^{r-s})\%2^r \ . \end{array} \tag{5}$$

In practice, since the addresses are r-bit long, there is no need to compute the modulo explicitly. It is sufficient to do the sum neglecting any possible overflow. Of course, the (3) is also replaced by

$$\text{IN} = (\text{OUT} + m2^{r-s})\%2^r \ . \tag{6}$$

3.2 Fractional-Length Delay Lines

It might be thought that, choosing a sufficiently high sampling rate, it is always possible to use delay lines having an integer number of samples. Actually, there are some good reasons that lead us to state that this is not the case in sound synthesis and processing.

In sound synthesis, the models have to be carefully tuned without resorting to very high sample rates. In particular, it is easy to verify that using integer-length delays in physical models we get errors in fundamental frequencies that

go well beyond the just noticeable difference in pitch[1] (see the appendix C). For instance, for a pressure wave propagating in air at normal temperature conditions, the spatial discretization given by the sampling rate $F_s = 44100Hz$ gives intervals of $0.0075m$, a distance that can produce well-perceivable pitch differences in a wind instrument.

Another reason for using fractional delays is that we often want to vary the delay lengths continuously, in order to reproduce effects such as glissando or vibrato. The adoption of integer-length delays would produce annoying discontinuities.

The most widely used techniques for implementing fractional delays are interpolation by FIR filters or by allpass filters. These two techniques are, in some sense, complementary. The choice of one of the two has to be made according to the peculiarities of the system to be simulated or of the architecture chosen for the implementation. In any case, a delay of length m is obtained by means of a delay line whose length is equal to the integer part of m, cascaded with a block capable to approximate a constant phase delay equal to the fractional part of m. We recall that the phase delay at a given frequency ω is the delay in time samples experienced by the sinusoidal component at frequency ω. For instance, consider a linear filtering block enclosed in a feedback loop (see sec. 3.4): the frequency of the k-th resonance f_k of the whole feedback system is found at the points where the phase response equates the multiples of 2π. At these frequencies, the components reappear in phase every round trip in the loop, thus reinforcing their amplitude at the output. The phase delay at frequency f_k is therefore the effective delay length at that frequency, that is the length of an ideal (linear phase) delay line that gives the same k-th resonance. Fig. 1 shows a phase curve and its crossings with multiples of 2π giving a distribution of resonances.

3.2.1 FIR Interpolation Filters

The easiest and most intuitive way to obtain a variable-length delay is to linearly interpolate the output of the line with the content of its preceding cell in the memory buffer. This corresponds to using the first-order FIR filter

$$H_l(z) = c_0 + c_1 z^{-1} \ . \tag{7}$$

[1] To figure this out, the reader can consider an m-sample delay line in a feedback loop. It gives a harmonic series of partials whose fundamental is $f_0 = \frac{F_s}{m}$ (see sec. 3.4). The set of integer delay lengths that give the best approximation to a tempered scale can be found and the curve of fundamental frequency errors can be drawn.

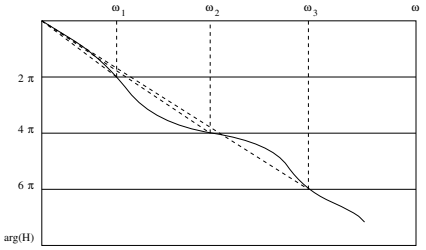

Figure 1: Graphical construction to find the series of resonances produced by a linear block in a feedback loop. The slope of the dashed lines indicates the phase delay at each resonance frequency.

Given a certain phase delay

$$\tau_{ph_0} = -\frac{1}{\omega_0} \arctan \frac{-c_1 \sin \omega_0}{c_0 + c_1 \cos \omega_0} \tag{8}$$

that has to be obtained at a given frequency ω_0, the following formulas give the coefficient values:

$$\begin{aligned} c_0 + c_1 &= 1 \\ c_1 &= \frac{1}{1 + \frac{\sin(\omega_0)}{\tan(\tau_{ph_0}\omega_0)} - \cos(\omega_0)} \approx \tau_{ph_0} \ , \end{aligned} \tag{9}$$

where the approximation is valid in the low-frequency range. The first of the (9) is needed in order to normalize the low-frequency response to one. In the special case that $c_0 = c_1 = \frac{1}{2}$ (averaging filter) the phase is linear and the delay is of half a sample. Unfortunately, the magnitude response of this interpolator is lowpass with a zero at the Nyquist frequency. Fig. 2 shows the magnitude, phase, and phase delay responses for several first-order linear interpolators. We can see that the phase is linear in most of the audio range, but the magnitude varies from the allpass to the lowpass with a zero at the Nyquist rate. When the interpolator is inserted within a feedback loop, its lowpass behavior can be treated as an additional frequency-dependent loss, which should be somewhat taken into account.

Interpolation filters can be of order higher than the first. We can do quadratic, cubic, or other polynomial interpolations. In general, the problem of designing an interpolator can be turned into the design of an l-th order FIR filter approximating a constant magnitude and linear phase frequency response. Several criteria can be adopted to drive the approximation problem. One approach is to impose that the first L derivatives of the error function will be zero

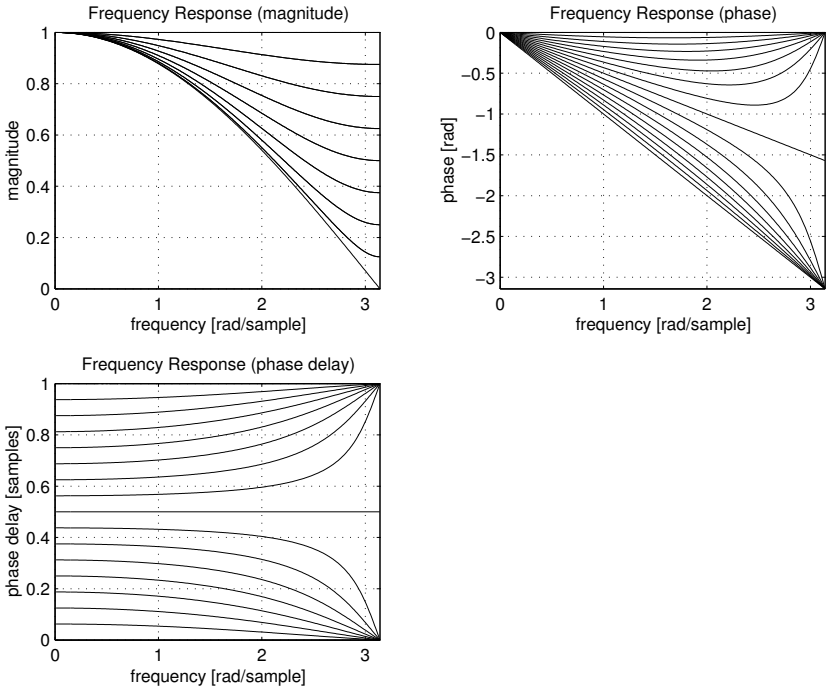

Figure 2: Magnitude, phase, and phase delay responses of a linear interpolation filter $(1-\alpha)+\alpha z^{-1}$ for $\alpha = k/16, k = 0, \ldots, 16$

at zero frequency. In this way we obtain maximally-flat filters whose coefficients are the same used in Lagrange interpolation as it is taught in numerical analysis courses. For a thorough treatment of interpolation filters we suggest reading the article [51]. Here we only point out that using high orders allows to keep the magnitude response close to unity and a phase response close to linear in a wide frequency band. Of course, this is paid in terms of computational complexity.

In special architectures, where the access to delay lines is governed by (5) and (6), the linear interpolation is implemented very efficiently by using the $r - s$ bits that are not used to access the 2^s-samples delay line. In fact, if the address is computed using r bits, the $r - s$ least significant bits represent the fractional part of the delay or, equivalenty, the coefficient c_1 of the interpolator.

Therefore, it is sufficient to access two consecutive delay cells and keep the values c_0 and $c_1 = 1 - c_0$ in two registers. The implementation of a glissando with these architectures is immediate and free from complications.

3.2.2 Allpass Interpolation Filters

Another widely used technique to obtain the fractional part of a desired delay length makes use of unit-magnitude IIR filters, i.e., allpass filters. Since the magnitude of these filters is constant there is no frequency-dependent attenuation, a property that can never be ensured by FIR filters. The simplest allpass filter has order one, and it has the following transfer function:

$$H_a(z) = \frac{c + z^{-1}}{1 + cz^{-1}} \,. \tag{10}$$

In order to make sure that the filter is stable, the coefficient c has to stay within the unit circle. Moreover, if we stick with real coefficients, c belongs to the real axis. The phase delay given by the filter (10) is shown in fig. 3 for several values of the coefficient c. It is clear that the phase delay is not as flat as in the case of the FIR interpolator, depicted in fig. 2.

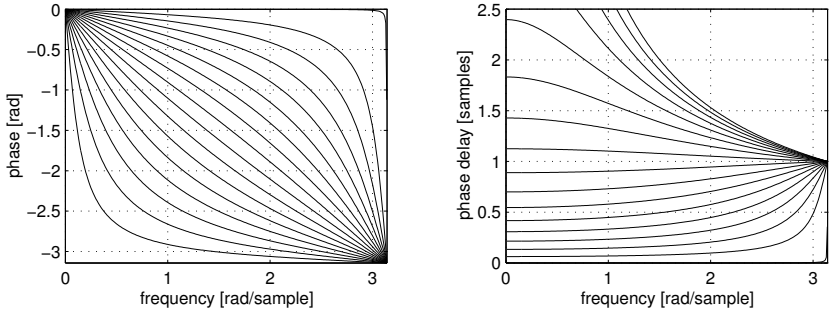

Figure 3: Phase response and phase delay of a first-order allpass filter for the values of the coefficient $c = 1.998k/17 - 0.999, k = 0, \ldots, 16$

It is easy to verify[2] that, at frequencies close to dc, the phase response of (10) takes the approximate form

$$\angle H(\omega) \approx -\frac{\sin(\omega)}{c + \cos(\omega)} + \frac{c \sin(\omega)}{1 + c\cos(\omega)} \approx -\omega \frac{1 - c}{1 + c} \,, \tag{11}$$

[2]The proof of (11) is left to the reader as a useful exercise.

where the first approximation is obtained by replacing the argument of the $\arctan$ with the function value and the second approximation, valid in an even smaller neighborhood, is obtained by approximating $\sin x$ with x and $\cos x$ with 1. The phase and group delay around dc are

$$\tau_{ph}(\omega) \approx \tau_{gr}(\omega) \approx \frac{1-c}{1+c}\,. \tag{12}$$

Therefore, the filter coefficient c can be easily determined from the desired low-frequency delay as

$$c = \frac{1-\tau_{ph}(0)}{1+\tau_{ph}(0)}\,. \tag{13}$$

Fig. 3 shows that the delay of the allpass filter is approximately constant only in a narrow frequency range. We can reasonably assume that such range, for positive values of c smaller than one, extends from 0 to $F_s/5$. With $F_s = 50$kHz we see that at $F_s/5 = 10$kHz we have an error of about 0.05 time samples. In a note at that frequency produced by a feedback delay line, such an error produces a pitch deviation smaller than 1%. For lower fundamental frequencies, such as those found in actual musical instruments, the error is smaller than the just noticeable difference measured with slow pitch modulations (see the appendix C).

If the first-order filter represents an elegant and efficient solution to the problem of tuning a delay line, it has also the relevant side effect of detuning the upper partials, due to the marked phase nonlinearity. Such detuning can be tolerated in most cases, but has to be taken into account in some other contexts. If a phase response closer to linear is needed, we can use higher-order allpass filters [51]. In some cases, especially in sound synthesis by physical modeling, a specific inharmonic distribution of resonances has to be approximated. This can be obtained by designing allpass filters that approximate a given phase response along the whole frequency axis. In these cases the problem of tuning is superseded by the more difficult problem of accurate partial positioning [83].

With allpass interpolators it is more complicated to handle continuous delay length variations, since the recursive structure of the filter does not show an obvious way of transferring memory cells from and to the delay line, as it was in the case of the FIR interpolator, which is constructed on the delay line by a certain number of taps. Indeed, the glissando can be implemented with the allpass filter by adding a new cell to the delay line whenever the filter coefficient becomes one and, at the same time, zeroing out the filter state variable and the coefficient. What is really more complicated with allpass filters is to handle sudden variations of the delay length, as they are found, for instance, when a

finger hole is opened in a wind instrument. In this case, the recursive nature of allpass filters causes annoying transients in the output signal. Ad hoc structures have been devised to cancel these transients [51].

3.3 The Non-Recursive Comb Filter

Sounds, propagating in the air, come into contact with surfaces and objects of various kinds and this interaction produces physical phenomena such as reflection, refraction, and diffraction. A simple and very important phenomenon is the reflection of sound about a planar surface. Due to a reflection such as this, a listener receives two delayed copies of the same signal. If the delay is larger than about a hundred milliseconds, the second copy is perceived as a distinguished echo, while if the delay is smaller than about ten milliseconds, the effect of a single reflection is perceived as a spectral coloration.

A simple model of single reflection can be constructed starting from the basic blocks described in this and in the preceding chapters. It is constructed as an m-samples delay line, with the incidental fractional part of m obtained by FIR interpolation or allpass filtering, cascaded with an attenuation coefficient g, possibly replaced by a filter if a frequency-dependent absorption has to be simulated. The output of this lossy delay line is summed to the direct signal. Let us analyze the structure in the case that m is integer and g is a positive constant not exceeding 1.

The difference equation is expressed as

$$y(n) = x(n) + g \cdot x(n - m) \ , \tag{14}$$

and, therefore, the transfer function is

$$H(z) = 1 + gz^{-m} \ . \tag{15}$$

In the case that $g = 1$, it is easy to see by using the De Moivre formula (see section A.6) that the frequency response of the comb filter has the following magnitude and group delay:

$$\begin{aligned} |H(\omega)| &= \sqrt{2(1 + \cos(\omega m))} \\ \tau_{gr,H}(\omega) &= \tfrac{m}{2} \ , \end{aligned} \tag{16}$$

and it is straightforward to verify that the frequency band ranging from dc to the Nyquist rate comprises m zeros (antiresonances), equally spaced by F_s/mHz.

The phase response[3] is piecewise linear with discontinuities of π at the odd multiples of $Fs/2m$.

If $g < 1$, it is easy to see that the amplitude of the resonances is

$$P = 1 + g \,, \tag{17}$$

while the amplitude of the points of minimum (halfway between contiguous resonances) is

$$V = 1 - g \,. \tag{18}$$

An important parameter of this filtering structure, called non-recursive comb filter (or FIR comb), is the peak-to-valley ratio

$$\frac{P}{V} = \frac{1+g}{1-g} \,. \tag{19}$$

Fig. 4 shows the response of a non-recursive comb filter having length $m =$ 11samples and a reflection attenuation $g = 0.9$. The shape of the frequency response justifies the name comb given to the filter.

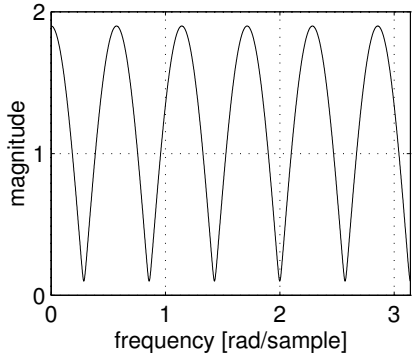

Figure 4: Magnitude of the frequency response of the comb FIR filter having coefficient $g = 0.9$ and delay length $m = 11$

The zeros of the comb filter are evenly distributed along the unit circle at the m-th roots of $-g$, as shown in figure 5.

[3]The reader is invited to calculate and plot the phase response.

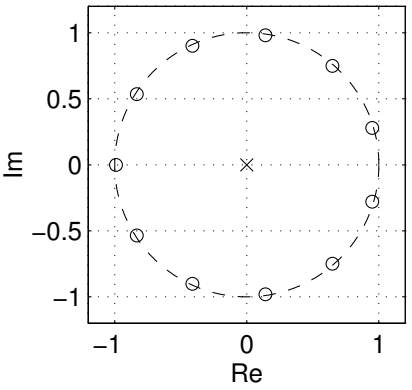

Figure 5: Zeros and poles of an FIR comb filter

3.4 The Recursive Comb Filter

A simple model of one-dimensional resonator can be constructed using the basic blocks presented in this and in the preceding chapters. It is composed by an m-samples delay line, with the incidental fractional part of m obtained by FIR interpolation or allpass filtering, in feedback loop with an attenuation coefficient g, possibly replaced by a filter in order to give different decay times at different frequencies. Let us analyze the whole filtering structure in the case that m is integer and g is a positive constant not exceeding 1.

The difference equation is expressed as

$$y(n) = x(n-m) + g \cdot y(n-m) \ , \tag{20}$$

and the transfer function is

$$H(z) = \frac{z^{-m}}{1 - gz^{-m}} \ . \tag{21}$$

Whenever $g < 1$, the stability is ensured. In the case that $g = 1$, the frequency response of the filter has the following magnitude and group delay:

$$\begin{array}{l} |H(\omega)| = \frac{1}{2\sin(\omega m/2)} \\ \tau_{gr,H}(\omega) = \frac{m}{2} \ , \end{array} \tag{22}$$

and it is easy to verify that the frequency band ranging from dc to the Nyquist rate comprises m vertical asymptotes (resonances), equally spaced by F_s/mHz.

If $g = 1$ the filter is at the limit of stability, and this is the only case when the phase response is piecewise linear[4], starting with the value $-\pi/2$ at dc, with discontinuities of π at the even multiples of $F_s/2m$.

If $g < 1$, it is easy to verify that the amplitude of the resonances is

$$P = \frac{1}{1-g} \,, \tag{23}$$

while the amplitude of the points of minimum (halfway between contiguous resonances) is

$$V = \frac{1}{1+g} \,. \tag{24}$$

An important parameter of this filtering structure, called recursive comb filter (or IIR comb), is the peak-to-valley ratio

$$\frac{P}{V} = \frac{1+g}{1-g} \,. \tag{25}$$

Fig. 6 shows the frequency response of a recursive comb filter having a delay line of $m = 11$ samples and feedback attenuation $g = 0.9$. The shape of the magnitude response justifies the name comb given to the filter.

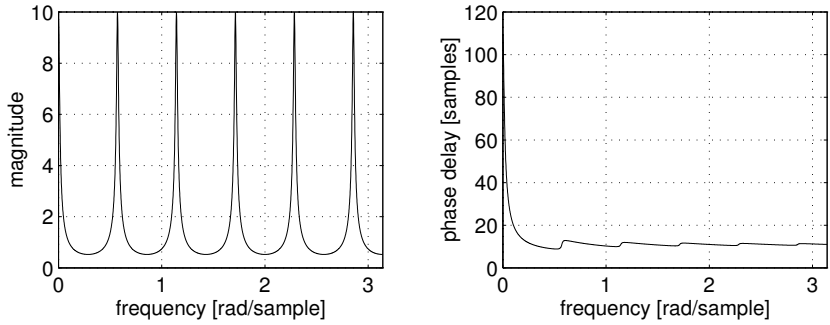

Figure 6: Magnitude and phase delay response of the recursive comb filter having coefficient $g = 0.9$ and delay length $m = 11$

The poles of the comb filter are evenly distributed along the unit circle at the m-th roots of g, as shown in figure 7.

[4]The reader is invited to calculate and plot the phase response.

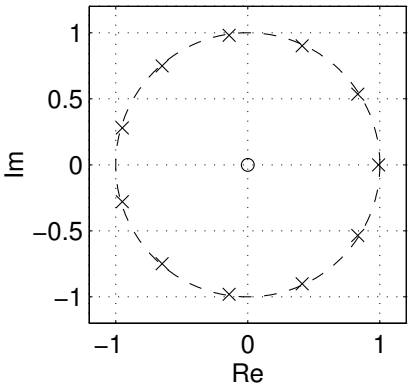

Figure 7: Zeros and poles of an IIR comb filter

In sound synthesis by physical modeling, a recursive comb filter can be interpreted as a simple model of lossy one-dimensional resonator, like a string, or a tube. This model can be used to simulate several instruments whose resonator is not persistently excited. In fact, if the input is a short burst of filtered noise, we obtain the basic structure of the plucked string synthesis algorithm due to Karplus and Strong [47].

3.4.1 The Comb-Allpass Filter

The filter given by the difference equation (20) has a frequency response characterized by evenly-distributed resonances. With a slight modification of its structure, such filter can be made allpass. In other words, the magnitude response of the filter can be made flat even though the impulse response remains almost the same (20). The modification is just a direct path connecting the input of the delay line to the filter output, as it is depicted in fig. 8. It is easy to

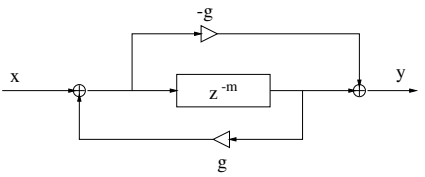

Figure 8: Allpass comb filter

see that the transfer function of the filter of fig. 8, called the allpass comb filter can be written as

$$H(z) = \frac{-g + z^{-m}}{1 - gz^{-m}} \ , \tag{26}$$

which has the structure of an allpass filter. It is interesting to note that the direct path introduces a nonzero sample at the time instant zero in the impulse response. All the following samples are just a scaled version of those of the impulse response of the comb filter, with a scaling factor equal to $1 - g^2$. The time properties, such as the time decay, are substantially unvaried. The allpass comb filter does not introduce any coloration in stationary signals. On the other hand, its effect is evident on signals exhibiting rapid transients, and for these signals we can not state that the filter is transparent.

3.5 Sound Effects Based on Delay Lines

Many of the effects commonly used in electroacoustic music are obtained by composition of time-varying delay lines, i.e., by lines whose length is modulated by slowly-varying signals. In order to avoid discontinuities in the signals, it is necessary to interpolate the delay lines in some way. The interpolation by means of allpass filters is applicable only for very slow modulations or for narrow-width modulations, since sudden changes in the state of allpass filters give rise to transients that can be perceived as signal distortions [30]. On the other hand, linear (or, more generally, polynomial) interpolation introduces frequency-dependent losses whose magnitude is dependent on the fractional length of the delay line. As the delay length is varied, these variable losses give an amplitude distortion due to amplitude modulation of the various frequency components. Coupled to amplitude modulation, there is also phase modulation due to phase nonlinearity of the interpolator, in both cases of FIR and IIR interpolation.

The terminology used for audio effects is not consistent, as terms such as flanger, chorus, and phaser are often associated with a large variety of effects, that can be quite different from each other. A flanger is usually defined as an FIR comb filter whose delay length is sinusoidally modulated between a minimum and a maximum value. This has the effect of expanding and contracting the harmonic series of notches of the frequency response. The name flanger derives from the old practice, used long ago in the analog recording studios, to alternatively slow down the speed of two tape recorders or two turntables playing the same music track by pressing a finger on the flanges.

The name phaser is most often reserved for structures similar to the comb FIR filter, with the difference that the notches are not harmonically distributed. Orfanidis [67] proposes to use, instead of the delay line, a bunch of parametric notch filters such as those presented in sec. 2.2.4. Each notch is controllable in its frequency position and width. Smith [96], instead, proposes to use a large allpass filter instead of the delay line. If this allpass filter is obtained as a cascade of second-order allpass sections, it becomes possible to control and modulate the position of any single pole couple, which represent all the single notches of the overall response. A common feature of flangers and phasers is the relatively large distance between the notches. Vice versa, if the notches are very dense, the term chorus is preferred. Orfanidis [67], suggests to implement a chorus as a parallel of FIR comb filters, where the delay lengths are randomly modulated around values that are slightly different from each other. This should simulate the deviations in time and height that are found in performances of a choir singing in unison. Vice versa, Dattorro [30] says that a chorus can be obtained by the same structure used for the flanger, with a difference that the delay lengths have to be set to larger values than for the flanger. In this way, the notches are made more dense. For the flanger the suggested nominal delay is 1msec and for the chorus it is 5msec. If the objective is to recreate the effect of a choir singing in unison, the fact of having many notches in the spectrum is generally disliked. Dattorro [30] proposes a partial solution that makes use of a recursive allpass filter, where the delay line is read by two pointers, one is kept fixed and produces the feedback signal, the other is varied to pick up the signal that is fed directly to the output. In this way, when both the pointers are at the nominal position, the structure does not introduce any coloration for stationary signals.

A final remark is reserved to the spatialization of these comb-based effects. In general, flanging, phasing, and chorusing effects can be obtained from two different time-varying allpass chains, whose outputs feed different loudspeakers. In this case, sums and subtractions between signals at the different frequencies happen “on air” in a way dependent from position. Therefore, the spatial sensation is largely due to the different spectral coloration found in different points of the listening area.

Exercise

The reader is invited to write a chorus/flanger based on comb or allpass comb filters using a language for sound processing (e.g., CSound). As an input signal, try a sine wave and a noisy signal. Then, implement a phaser by

cascading several first-order allpass filters having coefficients between 0 and 1.

3.6 Spatial sound processing

The spatial processing of sound is a wide topic that would require at least a thick book chapter on its own [82]. Here we only describe very briefly a few techniques for sound spatialization and reverberation. In particular, techniques for sound spatialization are different if the target display is by means of headphones or loudspeakers.

3.6.1 Spatialization

Spatialization with headphones

Humans can localize sound sources in a 3D space with good accuracy using several cues. If we can rely on the assumption that the listener receives the sound material via a stereo headphone we can reproduce most of the cues that are due to the filtering effect of the pinna–head–torso system, and inject the signal artificially affected by this filtering process directly to the ears.

Sound spatialization for headphones can be based on interaural intensity and time differences (see the appendix C). It is possible to use only one of the two cues, but using both cues will provide a stronger spatial impression. Of course, interaural time and intensity differences are just capable of moving the apparent azimuth of a sound source, without any sense of elevation. Moreover, the apparent source position is likely to be located inside the head of the listener, without any sense of externalization. Special measures have to be taken in order to push the virtual sources out of the head.

A finer localization can be achieved by introducing frequency-dependent interaural differences. In fact, due to diffraction the low frequency components are barely affected by IID, and the ITD is larger in the low frequency range. Calculations done with a spherical head model and a binaural model [49, 73] allow to draw approximated frequency-dependent ITD curves, one being displayed in fig. 9.a for 30^o of azimuth. The curve can be further approximated by constant segments, one corresponding to a delay of about 0.38ms in low frequency, and the other corresponding to a delay of about 0.26ms in high frequency. The low-frequency limit can in general be obtained for a general incident angle θ by the formula

$$\mathrm{ITD} = \frac{1.5\delta}{c} \sin\theta \ , \tag{27}$$

where δ is the inter-ear distance in meters and c is the speed of sound. The crossover point between high and low frequency is located around 1kHz. Similarly,

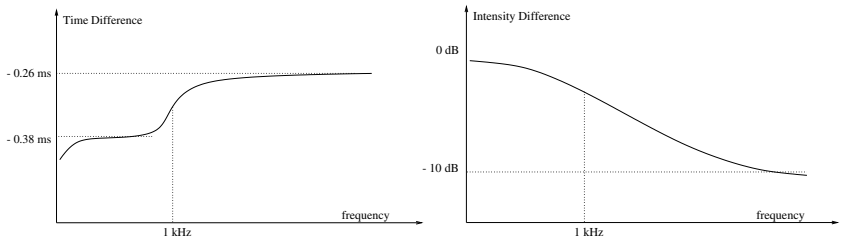

Figure 9: Frequency-dependent interaural time (a) and intensity (b) difference for azimuth 30^o.

the IID should be made frequency dependent. Namely, the difference is larger for high-frequency components, so that we have IID curves such as that reported in fig. 9.b for 30^o of azimuth. The IID and ITD are shown to change when the source is very close to the head [32]. In particular, sources closer than five times the head radius increase the intensity difference in low frequency. The ITD also increases for very close sources but its changes do not provide significant information about source range.

Several researchers have measured the filtering properties of the system pinna - head - torso by means of manikins or human subjects. A popular collection of measurements was taken by Gardner and Martin using a KEMAR dummy head, and made freely available [36, 38, 2]. Measurements of this kind are usually taken in an anechoic chamber, where a loudspeaker plays a test signal which invests the head from the desired direction. The directions should be taken in such a way that two neighbor directions never exceed the localization blur, which ranges from about $\pm 3^\circ$ in azimuth for frontal sources, to about $\pm 20^\circ$ in elevation for sources above and slightly behind the listener [13]. The result of the measurements is a set of Head-Related Transfer Functions (HRIR) that can be directly used as coefficients of a pair of FIR filters. Since the decay time of the HRIR is always less than a few milliseconds, 256 to 512 taps are sufficient at a sampling rate of 44.1kHz.

A cookbook of HRIRs and direct convolution seems to be a viable solution for providing directionality to sound sources using current technology. A fundamental limitation comes from the fact that HRIRs vary widely between different subjects, in such an extent that front-back reversals are fairly common

when listening through someone else's HRIRs. Using individualized HRIRs dramatically improves the quality of localization. Moreover, since we unconsciously use small head movements to resolve possible directional ambiguities, head-motion tracking is also desirable.

There are some reasons that make a model of the external hearing system more desirable than a raw catalog of HRIRs. First of all, a model might be implemented more efficiently, thus allowing more sources to be spatialized in real time. Second, if the model is well understood, it might be described with a few parameters having a direct relationship with physical or geometric quantities. This latter possibility can save memory and allow easy calibration.

Modeling the structural properties of the system pinna - head - torso gives us the possibility to apply continuous variation to the positions of sound sources and to the morphology of the listener. Much of the physical/geometric properties can be understood by careful analysis of the HRIRs, plotted as surfaces, functions of the variables time and azimuth, or time and elevation. This is the approach taken by Brown and Duda [19] who came up with a model which can be structurally divided into three parts:

- Head Shadow and ITD
- Shoulder Echo
- Pinna Reflections

Starting from the approximation of the head as a rigid sphere that diffracts a plane wave, the shadowing effect can be effectively approximated by a first-order continuous-time system, i.e., a pole-zero couple in the Laplace complex plane:

$$s_z = \frac{-2\omega_0}{\alpha(\theta)} \tag{28}$$

$$s_p = -2\omega_0 \ , \tag{29}$$

where ω_0 is related to the effective radius a of the head and the speed of sound c by

$$\omega_0 = \frac{c}{a} \ . \tag{30}$$

The position of the zero varies with the azimuth θ (see fig. 10 of the appendix C)) according to the function

$$\alpha(\theta) = 1.05 + 0.95 \cos\left(\frac{\theta - \theta_{\text{ear}}}{150^\circ} 180^\circ\right) , \tag{31}$$

where θ_{ear} is the angle of the ear that is being considered, typically 100° for the right ear and -100° for the left ear. The pole-zero couple can be directly translated into a stable IIR digital filter by bilinear transformation, and the resulting filter (with proper scaling) is

$$H_{\text{hs}} = \frac{(\omega_0 + \alpha F_s) + (\omega_0 - \alpha F_s) z^{-1}}{(\omega_0 + F_s) + (\omega_0 - F_s) z^{-1}} \ . \tag{32}$$

The ITD can be obtained by means of a first-order allpass filter [65, 100] whose group delay in seconds is the following function of the azimuth angle θ:

$$\tau_{\text{h}}(\theta) = \frac{a}{c} + \begin{cases} -\frac{a}{c}\cos\left(\theta - \theta_{\text{ear}}\right) & \text{if } 0 \le |\theta - \theta_{\text{ear}}| < \frac{\pi}{2} \\ \frac{a}{c}\left(|\theta - \theta_{\text{ear}}| - \frac{\pi}{2}\right) & \text{if } \frac{\pi}{2} \le |\theta - \theta_{\text{ear}}| < \pi \end{cases} \ . \tag{33}$$

Actually, the group delay provided by the allpass filter varies with frequency, but for these purposes such variability can be neglected. Instead, the filter (32) gives an excess delay at DC that is about 50% that given by (33). This increase of the group delay at DC is exactly what one observes for the real head [49], and it has already been outlined in fig. 9. The overall magnitude and group delay responses of the block responsible for head shadowing and ITD are reported in fig. 10.

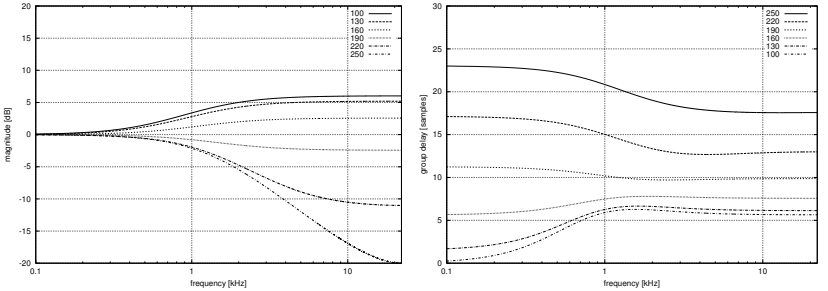

Figure 10: Magnitude and Group Delay responses of the block responsible for head shadowing and ITD ($F_s = 44100Hz$). Azimuth ranging from θ_{ear} to $\theta_{\text{ear}} + 150^\circ$.

In a rough approximation, the shoulder and torso effects are synthesized in a single echo. An approximate expression of the time delay can be deduced by the measurements reported in [19, fig. 8]

$$\tau_{\text{sh}} = 1.2\frac{180^\circ - \theta}{180^\circ}\left(1 - 0.00004\left((\phi - 80^\circ)\frac{180^\circ}{180^\circ + \theta}\right)^2\right) \text{ [msec]} \ , \tag{34}$$

where θ and ϕ are azimuth and elevation, respectively (see fig. 10 of the appendix C). The echo should also be attenuated as the source goes from frontal to lateral position.

Finally, the pinna provides multiple reflections that can be obtained by means of a tapped delay line. In the frequency domain, these short echoes translate into notches whose position is elevation dependent and that are frequently considered as the main cue for the perception of elevation [48]. A formula for the time delay of these echoes is given in [19].

The structural model of the pinna - head - torso system is depicted in Fig. 11 with all its three functional blocks, repeated twice for the two ears. The only difference in the two halves of the system is in the azimuth parameter that is θ for the right ear and $-\theta$ for the left ear.

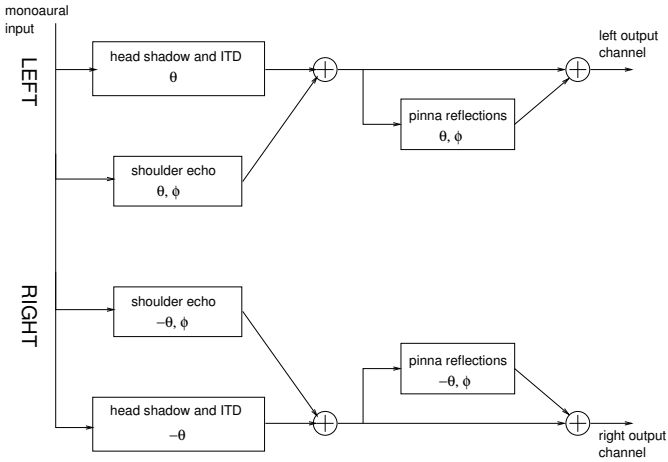

Figure 11: Structural model of the pinna - head - torso system

3D panning

The most popular and easy way to spatialize sounds using loudspeakers is amplitude panning. This approach can be expressed in matrix form for an arbitrary number of loudspeakers located at any azimuth though nearly equidistant from the listener. Such formulation is called Vector Base Amplitude Panning (VBAP) [72] and is based on a vector representation of positions in a Cartesian plane having its center in the position of the listener. In the two-loudspeaker

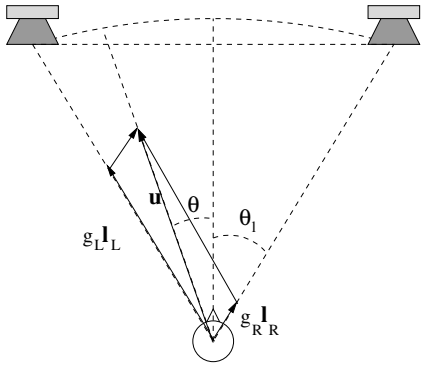

Figure 12: Stereo panning

case (figure 12), the unit-magnitude vector $\mathbf{u}$ pointing toward the virtual source can be expressed as a linear combination of the unit-magnitude column vectors $\mathbf{l}_L$ and $\mathbf{l}_R$ pointing toward the left and right loudspeakers, respectively. In matrix form, this combination can be expressed as

$$\mathbf{u} = \mathbf{L} \cdot \mathbf{g} = \begin{bmatrix} \mathbf{l}_L & \mathbf{l}_R \end{bmatrix} \begin{bmatrix} g_L \\ g_R \end{bmatrix}. \tag{35}$$

Except for degenerate loudspeaker positions, the linear system of equations (35) can be solved in the vector of gains $\mathbf{g}$. This vector has not, in general, unit magnitude, but can be normalized by appropriate amplitude scaling. The solution of system (35) implies the inversion of matrix $\mathbf{L}$, but this can be done beforehand for a given loudspeaker configuration.

The generalization to more than two loudspeakers in a plane is obtained by considering, at any virtual source position, only one couple of loudspeakers, thus choosing the best vector base for that position.

The generalization to three dimensions is obtained by considering vector bases formed by three independent vectors in space. The vector of gains for such a 3D vector base is obtained by solving the system

$$\mathbf{u} = \mathbf{L} \cdot \mathbf{g} = \begin{bmatrix} \mathbf{l}_L & \mathbf{l}_R & \mathbf{l}_Z \end{bmatrix} \begin{bmatrix} g_L \\ g_R \\ g_Z \end{bmatrix}. \tag{36}$$

Of course, having more than three loudspeakers in a 3D space implies, for any virtual source position, the selection of a local 3D vector base.

As indicated in [72], VBAP ensures maximum sharpness in sound source location. In fact:

- If the virtual source is located at a loudspeaker position only that loudspeaker has nonzero gain;
- If the virtual source is located on a line connecting two loudspeakers only those two loudspeakers have nonzero gain;
- If the virtual source is located on the triangle delimited by three adjacent loudspeakers only those three loudspeakers have nonzero gain.

The formulation of VBAP given here is consistent with the low frequency formulation of directional psychoacoustics. The extension to high frequencies have been also proposed with the name Vector Base Panning (VBP) [68].

Room within a room

A different approach to spatialization using loudspeakers can be taken by controlling the relative time delay between the loudspeaker feeds. A model supporting this approach was introduced by Moore [60], and can be described as a physical and geometric model. The metaphor underlying the Moore model is that of the Room within a Room, where the inner room has holes in the walls, corresponding to the positions of loudspeakers, and the outer room is the virtual room where sound events have to take place (fig. 13). The simplest form of

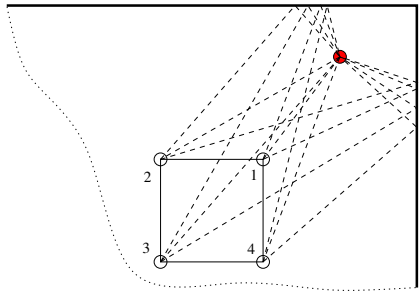

Figure 13: Moore's Room in a Room Model

spatialisation is obtained by drawing direct sound rays from the virtual sound source to the holes of the inner room. If the outer room is anechoic these are the only paths taken by sound waves to reach the inner room. The loudspeakers

will be fed by signals delayed by an amount proportional to the length of these paths, and attenuated according to relationship of inverse proportionality valid for propagation of spherical waves. In formulas, if l_i is the path length from the source to the i-th loudspeaker, and c is the speed of sound in air, the delay in seconds is set to

$$d_i = l_i/c \,, \tag{37}$$

and the gain is set to

$$g_i = \begin{cases} \frac{1}{l_i}, & l_i > 1 \\ 1, & l_i < 1 \end{cases} \quad . \tag{38}$$

The formula for the amplitude gain is such that sources within the distance of 1m from the loudspeaker[5] will be stuck to unity gain, thus avoiding the asymptotic divergence in amplitude implied by a point source of spherical waves.

The model is as accurate as the physical system being modeled would permit. A listener within a room would have a spatial perception of the outside soundscape whose accuracy will increase with the number of windows in the walls. Therefore, the perception becomes sharper by increasing the number of holes/loudspeakers. Indeed, some of the holes will be masked by some walls, so that not all the rays will be effective [6] (e.g. the rays to loudspeaker 3 in fig. 13). In practice, the directional clarity of spatialisation is increased if some form of directional panning is added to the base model, so that loudspeakers opposite to the direction of the sound source are severely attenuated. With this trick, it is not necessary to burden the model with an algorithm of ray-wall collision detection.

The Moore model is suitable to provide consistent and robust spatialization to extended audiences [60]. A reason for robustness might be found in the fact that simultaneous level and time differences are applied to the loudspeakers. This has the effect to increase the lateral displacement [13] even for virtual sources such that the rays to different loudspeaker have similar lengths. Indeed, the localization of the sound source gets even sharper if the level control is driven by laws that roll off more rapidly than the physical $1/d$ law of spherical waves. In practical realizations, the best results are obtained by tuning the model after psychophysical experimentation [54].

An added benefit of the Room within a Room model is that the Doppler effect is intrinsically implemented. As the virtual sound source is moved in the outer room the delay lines representing the virtual rays change their lengths, thus producing the correct pitch shifts. It is true that different transpositions

[5] This distance is merely conventional.

[6] We are neglecting diffraction from this reasoning.

might affect different loudspeakers, as the variations are different for different rays, but this is consistent with the physical robustness of the technique.

The model of the Room within a Room works fine if the movements of the sound source are confined to a virtual space external to the inner room. This corresponds to an enlargement of the actual listening space and it is often a highly desirable situation. Moreover, it is natural to model the physical properties of the outer room, adding reflections at the walls and increasing the number of rays going from a sound source to the loudspeakers. This configuration, illustrated in fig. 13 with first-order reflections, is a step from spatialization to reverberation.

3.6.2 Reverberation

Classic reverberation tools

In the second half of the twentieth century, several engineers and acousticians tried to invent electronic devices capable to simulate the long-term effects of sound propagation in enclosures [14]. The most important pioneering work in the field of artificial reverberation has been that of Manfred Schroeder at the Bell Laboratories in the early sixties [88, 89, 90, 91, 93]. Schroeder introduced the recursive comb filters (section 3.4) and the delay-based allpass filters (section 3.4.1) as computational structures suitable for the inexpensive simulation of complex patterns of echoes. These structures rapidly became standard components used in almost all the artificial reverberators designed until nowadays [61]. It is usually assumed that the allpass filters do not introduce coloration in the input sound. However, this assumption is valid from a perceptual viewpoint only if the delay line is much shorter than the integration time of the ear, i.e. about 50ms [111]. If this is not the case, the time-domain effects become much more relevant and the timbre of the incoming signal is significantly affected.

In the seventies, Michael Gerzon generalized the single-input single-output allpass filter to a multi-input multi-output structure, where the delay line of m samples has been replaced by a order-N unitary network [40]. Examples of trivial unitary networks are orthogonal matrices, parallel connections of delay lines, or allpass filters. The idea behind this generalization is that of increasing the complexity of the impulse response without introducing appreciable coloration in frequency. According to Gerzon's generalization, allpass filters can be nested within allpass structures, in a telescopic fashion. Such embedding is shown to be equivalent to lattice allpass structures [39], and it is realizable as

long as there is at least one delay element in the block $A(z)$, which replaces the delay line in fig. 8.

An extensive experimentation on structures for artificial reverberation was conducted by Andy Moorer in the late seventies [61]. He extended the work done by Schroeder [90] in relating some basic computational structures (e.g., tapped delay lines, comb and allpass filters) with the physical behavior of actual rooms. In particular, it was noticed that the early reflections have great importance in the perception of the acoustic space, and that a direct-form FIR filter can reproduce these early reflections explicitly and accurately. Usually this FIR filter is implemented as a tapped delay line, i.e. a delay line with multiple reading points that are weighted and summed together to provide a single output. This output signal feeds, in Moorer's architecture, a series of allpass filters and a parallel of comb filters(see fig. 14) . Another improvement introduced by Moorer was the replacement of the simple gain of feedback delay lines in comb filters with lowpass filters resembling the effects of air absorption and lossy reflections.

The construction of high-quality reverberators is half an art and half a science. Several structures and many parameterizations were proposed in the past, especially in non-disclosed form within commercial reverb units [29]. In most cases, the various structures are combinations of comb and allpass elementary blocks, as suggested by Schroeder in the early works. As an example, we look more carefully at the Moorer's preferred structure [61], depicted in fig.14. The block (a) takes care of the early reflections by means of a tapped delay line. The resulting signal is forwarded to the block (b), which is the parallel of a direct path on one branch, and a delayed, attenuated diffuse reverberator on the other branch. The output of the reverberator is delayed in such a way that the last of the early echoes coming out of block (a) reaches the output before the first of the non-null samples coming out of the diffuse reverberator. In Moorer's preferred implementation, the reverberator of block (b) is best implemented as a parallel of six comb filters, each with a first-order lowpass filter in the loop, and a single allpass filter. In [61], it is suggested to set the allpass delay length to 6ms and the allpass coefficient to 0.7. Despite the fact that any allpass filter does not add coloration in the magnitude frequency response, its time response can give a metallic character to the sound, or add some unwanted roughness and granularity. The feedback attenuation coefficients g_i and the lowpass filters of the comb filters can be tuned to resemble a realistic and smooth decay. In particular, the attenuation coefficients g_i determine the overall decay time of the series of echoes generated by each comb filter. If the desired decay time (usually defined for an attenuation level of 60dB) is T_d, the gain of each comb

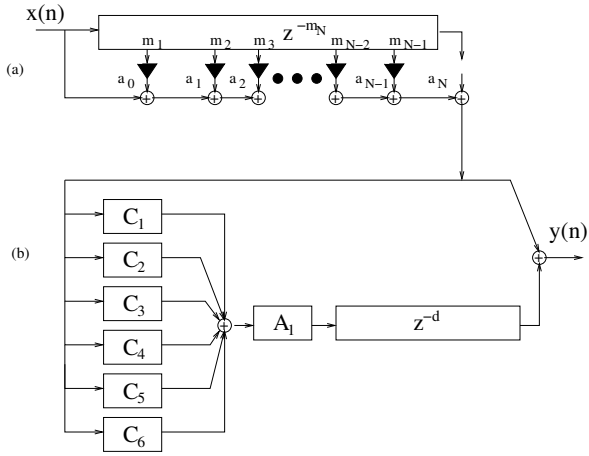

Figure 14: Moorer's reverberator

filter has to be set to

$$g_i = 10^{-3\frac{m_i}{T_d F_s}} \ , \tag{39}$$

where F_s is the sample rate and m_i is the delay length in samples. Further attenuation at high frequencies is provided by the feedback lowpass filters, whose coefficient can also be related with decay time at a specific frequency or fine tuned by direct experimentation. In [61], an example set of feedback attenuation and allpass coefficients is provided, together with some suggested values of the delay lengths of the comb filters. As a rule of thumb, they should be distributed over a ratio $1 : 1.5$ between 50 and 80ms. Schroeder suggested a number-theoretic criterion for a more precise choice of the delay lengths [91]: the lengths in samples should be mutually coprime (or incommensurate) to reduce the superimposition of echoes in the impulse response, thus reducing the so called flutter echoes. This same criterion might be applied to the distances between each echo and the direct sound in early reflections. However, as it was noticed by Moorer [61], the results are usually better if the taps are positioned according to the reflections computed by means of some geometric modeling technique, such as the image method [3, 18]. Indeed, even the lengths of the recirculating delays can be computed from the geometric analysis of the normal modes of actual room shapes.

Feedback Delay Networks

In 1982, J. Stautner e M. Puckette [101] introduced a structure for artificial reverberation based on delay lines interconnected in a feedback loop by means of a matrix (see fig. 15). Later, structures such as this have been called Feedback Delay Networks (FDNs). The Stautner-Puckette FDN was obtained as a vector generalization of the recursive comb filter (20), where the m-sample delay line was replaced by a bunch of delay lines of different lengths, and the feedback gain g was replaced by a feedback matrix $\mathbf{G}$. Stautner and Puckette proposed the following feedback matrix:

$$\mathbf{G} = g \begin{bmatrix} 0 & 1 & 1 & 0 \\ -1 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{bmatrix} / \sqrt{2} \, . \tag{40}$$

Due to its sparse special structure, $\mathbf{G}$ requires only one multiply per output channel.

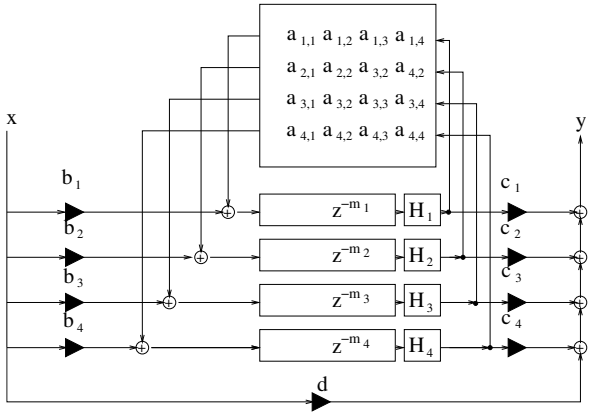

Figure 15: Fourth-order Feedback Delay Network

More recently, Jean-Marc Jot investigated the possibilities of FDNs very thoroughly. He proposed to use some classes of unitary matrices allowing efficient implementation. Moreover, he showed how to control the positions of the poles of the structure in order to impose a desired decay time at various frequencies [44]. His considerations were driven by perceptual criteria with the

general goal to obtain an ideal diffuse reverb. In this context, Jot introduced the important design criterion that all the modes of a frequency neighborhood should decay at the same rate, in order to avoid the persistence of isolated, ringing resonances in the tail of the reverb [45]. This is not what happens in real rooms though, where different modes of close resonance frequencies can be differently affected by wall absorption [63]. However, it is generally believed that the slow variation of decay rates with frequency produces smooth and pleasant impulse responses.

Referring to fig. 15, an FDN is built starting from N delay lines, each being $\tau_i = m_i T_s$ seconds long, where $T_s = 1/F_s$ is the sampling interval. The FDN is completely described by the following equations:

$$\begin{aligned} y(n) &= \sum_{i=1}^{N} c_i s_i(n) + dx(n) \\ s_i(n+m_i) &= \sum_{j=1}^{N} a_{i,j} s_j(n) + b_i x(n) \end{aligned} \tag{41}$$

where $s_i(n)$, $1 \leq i \leq N$, are the delay outputs at the n-th time sample. If $m_i = 1$ for every i, we obtain the well known state space description of a discrete-time linear system [46]. In the case of FDNs, m_i are typically numbers on the orders of hundreds or thousands, and the variables $s_i(n)$ are only a small subset of the system state at time n, being the whole state represented by the content of all the delay lines.

From the state-variable description of the FDN it is possible to find the system transfer function [80, 84] as

$$H(z) = \frac{Y(z)}{X(z)} = \mathbf{c}^T [\mathbf{D}(z^{-1}) - \mathbf{A}]^{-1} \mathbf{b} + d. \tag{42}$$

The diagonal matrix $\mathbf{D}(z) = \text{diag}\,(z^{-m_1}, z^{-m_2}, \ldots z^{-m_N})$ is called the delay matrix, and $\mathbf{A} = [a_{i,j}]_{N \times N}$ is called the feedback matrix.

The stability properties of a FDN are all ascribed to the feedback matrix. The fact that $\|A\|^n$ decays exponentially with n ensures that the whole structure is stable [80, 84].

The poles of the FDN are found as the solutions of

$$\det[\mathbf{A} - \mathbf{D}(z^{-1})] = 0\ . \tag{43}$$

In order to have all the poles on the unit circle it is sufficient to choose a unitary matrix. This choice leads to the construction of a lossless prototype but this is not the only choice allowed.

In practice, once we have constructed a lossless FDN prototype, we must insert attenuation coefficients and filters in the feedback loop (blocks G_i in figure 15). For instance, following the indications of Jot [45], we can cascade every delay line with a gain

$$g_i = \alpha^{m_i} \ . \tag{44}$$

This corresponds to replacing $D(z)$ with $D(z/\alpha)$ in (42). With this choice of the attenuation coefficients, all the poles are contracted by the same factor α. As a consequence, all the modes decay with the same rate, and the reverberation time (defined for a level attenuation of 60dB) is given by

$$T_d = \frac{-3T_s}{\log \alpha} \ . \tag{45}$$

In order to have a faster decay at higher frequencies, as it happens in real enclosures, we must cascade the delay lines with lowpass filters. If the attenuation coefficients g_i are replaced by lowpass filters, we can still get a local smoothness of decay times at various frequencies by satisfying the condition (44), where g_i and α have been made frequency dependent:

$$G_i(z) = A^{m_i}(z), \tag{46}$$

where $A(z)$ can be interpreted as per-sample filtering [43, 45, 98].

It is important to notice that a uniform decay of neighbouring modes, even though commonly desired in artificial reverberation, is not found in real enclosures. The normal modes of a room are associated with stationary waves, whose absorption depends on the spatial directions taken by these waves. For instance, in a rectangular enclosure, axial waves are absorbed less than oblique waves [63]. Therefore, neighbouring modes associated with different directions can have different reverberation times. Actually, for commonly-found rooms having irregularities in the geometry and in the materials, the response is close to that of a room having diffusive walls, where the energy rapidly spreads among the different modes. In these cases, we can find that the decay time is quite uniform among the modes [50].

The most delicate part of the structure is the feedback matrix. In fact, it governs the stability of the whole structure. In particular, it is desirable to start with a lossless prototype, i.e. a reference structure providing an endless, flat decay. The reader interested in general matrix classes that might work as prototypes is deferred to the literature [44, 84, 81, 39]. Here we only mention the

class of circulant matrices, having general form [7]

$$\mathbf{A} = \begin{bmatrix} a(0) & a(1) & \dots & a(N-1) \\ a(N-1) & a(0) & \dots & a(N-2) \\ \dots & & & \\ a(1) & \dots & a(N-1) & a(0) \end{bmatrix}.$$

The stability of a FDN is related to the magnitude of its eigenvalues, which can be computed by the Discrete Fourier Transform of the first raw, in the case of a circulant matrix. By keeping these eigenvalues on the unit circle (i.e., magnitude one) we ensure that the whole structure is stable and lossless. The control over the angle of the eigenvalues can be translated into a direct control over the degree of diffusion of the enclosure that is being simulated by the FDN. The limiting cases are the diagonal matrix, corresponding to perfectly reflecting walls, and the matrix whose rows are sequences of equal-magnitude numbers and (pseudo-)randomly distributed signs [81].

Another critical set of parameters is given by the lengths of the delay lines. Several authors suggested to use lengths in samples that are mutually coprime numbers in order to minimize the collision of echoes in the impulse response. However, if the FDN is linked to a physical and geometrical interpretation, as it is done in the Ball-within-the-Box model [79], the delay lengths are derived from the geometry of the room being simulated and the resulting digital reverb quality is related to the quality of the actual room. In the case of a rectangular room, a delay line will be associated to a harmonic series of normal modes, all obtainable from a plane wave loop that bounces back and forth within the enclosure.

Convolution with Room Impulse Responses

If the impulse response of a target room is readily available, the most faithful reverberation method would be to convolve the input signal with such a response. Direct convolution can be done by storing each sample of the impulse response as a coefficient of an FIR filter whose input is the dry signal. Direct convolution becomes easily impractical if the length of the target response exceeds small fractions of a second, as it would translate into several hundreds of taps in the filter structure. A solution is to perform the convolution block by block in the frequency domain: Given the Fourier transform of the impulse response, and the Fourier transform of a block of input signal, the two

[7] A matrix such as this is used in the Csound `babo` opcode.

can be multiplied point by point and the result transformed back to the time domain. As this kind of processing is performed on successive blocks of the input signal, the output signal is obtained by overlapping and adding the partial results [65]. Thanks to the FFT computation of the discrete Fourier transform, such technique can be significantly faster. A drawback is that, in order to be operated in real time, a block of N samples must be read and then processed while a second block is being read. Therefore, the input-output latency in samples is twice the size of a block, and this is not tolerable in practical real-time environments.

The complexity–latency tradeoff is illustrated in fig. 16, where the direct-form and the block-processing solutions can be located, together with a third efficient yet low-latency solution [37, 64]. This third realization of convolution is based on a decomposition of the impulse response into increasingly-large chunks. The size of each chunk is twice the size of its predecessor, so that the latency of prior computation can be occupied by the computations related to the following impulse-response chunk.

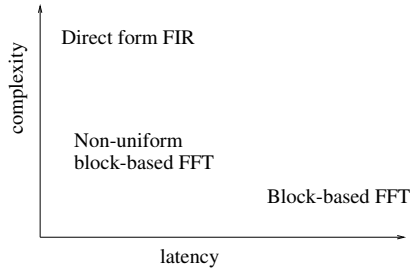

Figure 16: Complexity Vs. Latency tradeoff in convolution

Even if we have enough computer power to compute convolutions by long impulse responses in real time, there are still serious reasons to prefer reverberation algorithms based on feedback delay networks in many practical contexts. The reasons are similar to those that make a CAD description of a scene preferable to a still picture whenever several views have to be extracted or the environment has to be modified interactively. In fact, it is not easy to modify a room impulse response to reflect some of the room attributes, e.g. its high-frequency absorption, and it is even less obvious how to spatialize the echoes of the impulse response in order to get a proper sense of envelopment. If the impulse response is coming from a spatial rendering algorithm, such as ray tracing, these manipulations can be operated at the level of room description,

and the coefficients of the room impulse response transmitted to the real-time convolver. In the low-latency block based implementations of convolution, we can even have faster update rates for the smaller early chunks of the impulse response, and slower update rates for the reverberant tail. Still, continuous variations of the room impulse response are easier to be rendered using a model of reverberation operating on a sample-by-sample basis.

Chapter 4

Sound Analysis

Sounds are time-varying signals in the real world and, indeed, all of their meaning is related to such time variability. Therefore, it is interesting to develop sound analysis techniques that allow to grasp at least some of the distinguished features of time-varying sounds, in order to ease the tasks of understanding, comparison, modification, and resynthesis.

In this chapter we present the most important sound analysis techniques. Special attention is reserved on criteria for choosing the analysis parameters, such as window length and type.

4.1 Short-Time Fourier Transform

The Short-Time Fourier Transform (STFT) is nothing more than Fourier analysis performed on slices of the time-domain signal. In order to slightly simplify the formulas, we are going to present the STFT under the assumption of unitary sample rate ($F_s = T^{-1} = 1$).

There are two complementary views of STFT: the filterbank view, and the DFT-based view.

4.1.1 The Filterbank View

Assume we have a prototype ideal lowpass filter, whose frequency response is depicted in fig. 1. Let $w(\cdot)$ and $W(\cdot)$ be the impulse response and transfer function, respectively, of such prototype filter.

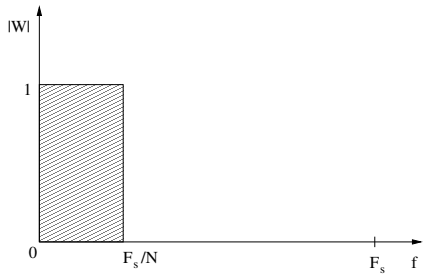

Figure 1: Frequency response of a prototype lowpass filter

We define modulation of a signal $y(n)$ by a carrier signal $e^{j\omega_0 n}$ as the (complex) multiplication $y(n)e^{j\omega_0 n}$. This translates, in the frequency domain, into a frequency shift by $\Delta\omega = \omega_0$ (shift theorem 1.2 of chapter 1). In other words, modulating a signal means moving its low frequency content onto an area around the carrier frequency. On the other hand, we call demodulation of a signal $y(n)$ its multiplication by $e^{-j\omega_0 n}$, that brings the components around ω_0 onto a neighborhood of dc.

By demodulation we can obtain a filterbank that slices the spectrum (between 0Hz and F_s) in N equal non-overlapping portions. Namely, we can translate the input signal in frequency and filter it by means of the prototype lowpass filter in order to isolate a specific slice of the frequency spectrum. This procedure is reported in fig. 2.

4.1.2 The DFT View

The scheme of fig. 2 can be obtained by Fourier transformation of a "windowed" sequence. We recall from section 1.3 that the DTFT of an infinite sequence is

$$Y(\omega) = \sum_{n=-\infty}^{+\infty} y(n)e^{-j\omega n} \; . \tag{1}$$

If the DTFT is computed on a portion of $y(\cdot)$, weighted by an analysis

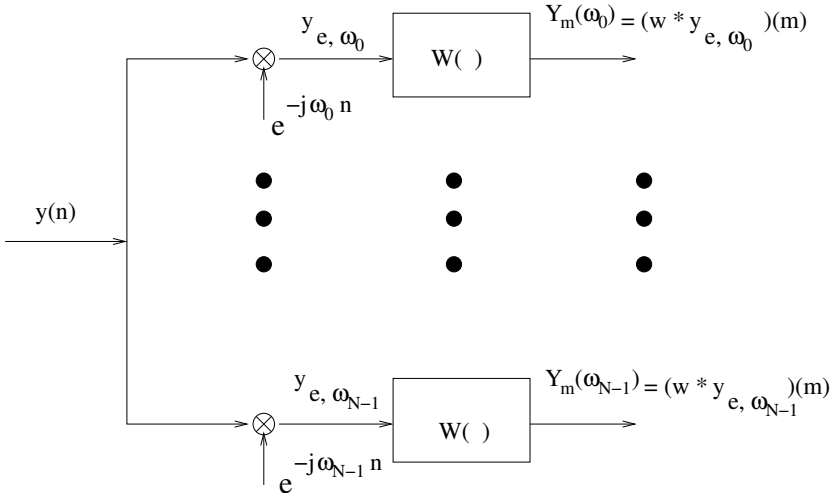

Figure 2: Decomposition of a signal into a set of non-overlapping frequency slices. $\omega_0, \ldots, \omega_{N-1}$ are the central frequencies of the bands of the analysis channels.

window $w(m-n)$, we get a frame of the STFT:

$$\begin{aligned} Y_m(\omega) &= \sum_{n=-\infty}^{+\infty} w(m-n)y(n)e^{-j\omega n} = \\ &= e^{-j\omega m} \sum_{r=-\infty}^{+\infty} w(r)y(m-r)e^{j\omega r} \ , \end{aligned} \tag{2}$$

where the third member of the equality is obtained by defining $r \stackrel{\triangle}{=} m-n$, and m is a variable accounting for the temporal dislocation of the window. Therefore, the STFT turns out to be a function of two variables, one can be thought of as frequency, the other is essentially a time shift.

The DTFT is a periodic function of a continuous variable, and it can be inverted by means of an integral computed over a period

$$w(m-n)y(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} Y_m(\omega)e^{j\omega n} d\omega \ . \tag{3}$$

By a proper alignment of the window ($m = n$) we can compute, if $w(0) \neq 0$

$$y(n) = \frac{1}{2\pi w(0)} \int_{-\pi}^{\pi} Y_n(\omega) e^{j\omega n} d\omega \; . \tag{4}$$

The STFT in its formulation (2) can be seen as convolution

$$Y_m(\omega) = (w * y_e)(m) \; , \tag{5}$$

where $y_e(n) = y(n)e^{-j\omega n}$ is the demodulated signal. If w is set to the impulse response of the ideal lowpass filter, and if we set $\omega = \omega_k$, we get a channel of the filterbank of fig. 2. In general, $w(\cdot)$ will be the impulse response of a non-ideal lowpass filter, but the filterbank view will keep its validity.

In practice, we need to compute the STFT on a finite set of N points. In what follows we assume that the window is $R \leq N$ samples long, so that we can use the DFT on N points, thus obtaining a sampling of the frequency axis between 0 and 2π in multiples of $2\pi/N$.

The k-th point in the transform domain (said the k-th bin of the DFT) is given by

$$Y_m(k) = \sum_{n=0}^{N-1} w(m-n) y(n) e^{-j\frac{2\pi kn}{N}} \tag{6}$$

and, by means of an inverse DFT

$$w(m-n)y(n) = \frac{1}{N} \sum_{k=0}^{N-1} Y_m(k) e^{j\frac{2\pi kn}{N}} \; . \tag{7}$$

By a proper alignment of the window ($m = n$), and assuming that $w(0) \neq 0$ we get

$$y(n) = \frac{1}{Nw(0)} \sum_{k=0}^{N-1} Y_n(k) e^{\frac{j2\pi kn}{N}} \; . \tag{8}$$

More generally, we can reconstruct (resynthesis) the time-domain signal by means of

$$y(n) = \frac{1}{Nw(m-n)} \sum_{k=0}^{N-1} Y_m(k) e^{\frac{j2\pi kn}{N}} \; , \tag{9}$$

where $w(m-n) \neq 0$, which is true, given an integer n_0, for a non-trivial window defined for

$$m + n_0 \leq n \leq m + n_0 + R - 1 \; . \tag{10}$$

Example

Figure 3 illustrates the operations involved in analysis and resynthesis of a frame of STFT ($R = 5$, $N = 8$). Reconstruction is possible for $1 \leq n \leq 5$ ($n_0 = -2$).

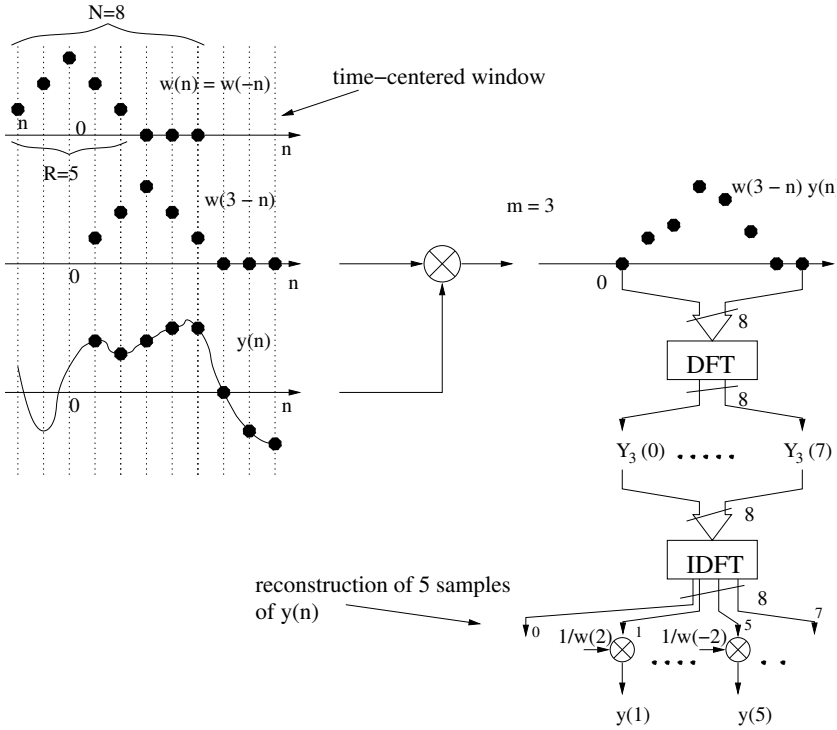

Figure 3: Analysis and resynthesis of a frame of STFT.

4.1.3 Windowing

The rectangular window

The simplest analysis window is the rectangular window

$$w_R(n) = \begin{cases} 1 & n = 0, \ldots, R-1 \\ 0 & elsewhere \end{cases}, \tag{11}$$

Considered a filter having (11) as its impulse response, the frequency response is found by Fourier-transformation of $w_R(n)$:

$$W_R(\omega) = \sum_{n=-\infty}^{+\infty} w_R(n)e^{-j\omega n} = \sum_{n=0}^{R-1} e^{-j\omega n} = \frac{1-e^{-j\omega R}}{1-e^{-j\omega}} = \stackrel{\triangle}{=} \text{sinc}_R(\omega) = e^{-j\omega\frac{R-1}{2}} \frac{\sin\frac{\omega R}{2}}{\sin\frac{\omega}{2}} . \quad (12)$$

The real part of the function $\text{sinc}_R(\omega)$ is plotted in figure 4 for different values of the window length R.

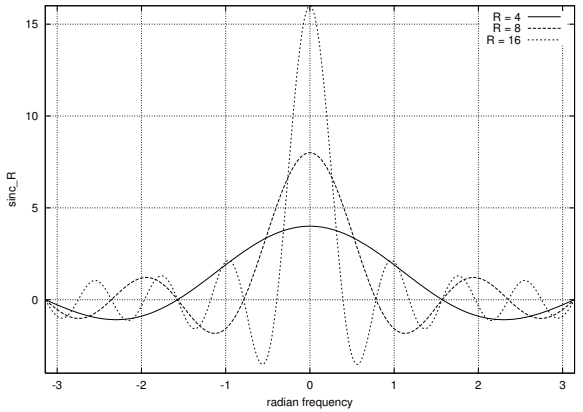

Figure 4: $\text{sinc}_R(\omega)$ for different values of window length R.

In figure 4, it can be noticed that $2\pi/R$ is the zero closest to dc. Therefore, we can say that if we use the rectangular window as a prototype of filter represented in figure (2), the equivalent bandwidth is $2\pi/R$. If we neglect aliasing for a moment, we realize that we can decimate each channel $Y_m(\omega_k)$ by a factor R without loosing any information.

A superficial look at the expression (12) seems to indicate that the shifted replicas of sinc_R produce aliasing in the base band $\left(-\frac{2\pi}{R}, \frac{2\pi}{R}\right)$. Indeed, if we sum R shifted replicas we verify that the aliasing components cancel out. Therefore, with this window, it is possible to decimate the output channels by a factor equal to the window length. Furthermore, if we choose $N = R$, we can perform one FFT per frame and advance the window by N samples at each step.

According to (7), the reconstruction (resynthesis) of the analyzed signal can be obtained by filterbank summation, as depicted in figure 5. The reconstruction can be interpreted as a bank of oscillators driven by the analysis data. The two stages represented in figures 2 and 5, taken as a whole, are often called the phase vocoder.

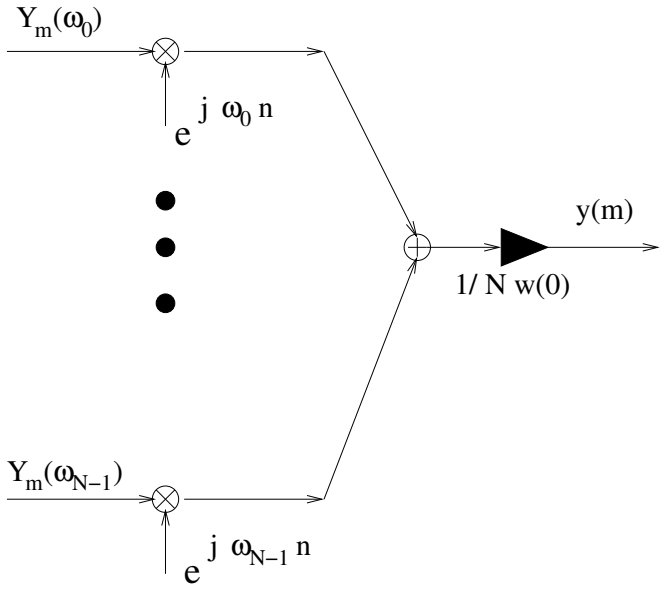

Figure 5: Reconstruction of a signal from a set of non-overlapping frequency slices. $\omega_0, \ldots, \omega_{N-1}$ are the central frequencies of the bands of the analysis channels.

Between the analysis stage of figure 2 and the synthesis stage of figure 5, a decimation stage can be inserted. Namely, with the rectangular window we can reduce the intermediate sampling rate down to F_s/R. Of course, in order to do the filter bank summation of figure 5, an interpolation stage will be needed to take the sampling rate back to F_s.

For the rectangular window, the window is shifted in time by R samples after each DFT computation. This temporal shift is technically called hop size. In the case of the rectangular window, hop sizes smaller than R do not add any information to the analysis.

Commonly-used windows

In practice, signal analysis is seldom performed using rectangular windows, because its frequency response has side lobes that are significantly high thus potentially inducing erroneous estimations of frequency components. In general, there is a tradeoff between the main-lobe width and the side-lobe level that can be exploited by choosing or designing an appropriate window. Table 4.1 describes concisely the form and features of the most-commonly used analysis windows.

Window Name	$w(n)$ in $-\frac{R-1}{2} \leq n \leq \frac{R-1}{2}$	Main-lobe Width ($\times \frac{\pi}{R}$)	Side-lobe Level [dB]
Rectangular	1	4	-13.3
Hann	$\frac{1}{2}\left(1+\cos\left(\frac{2\pi n}{R}\right)\right)$	8	-31.5
Hamming	$0.54+0.46\cos\frac{2\pi n}{R}$	8	-42.7
Blackman	$0.42 + 0.5\cos\frac{2\pi n}{R} + 0.08\cos\frac{4\pi n}{R}$	12	-58.1

Table 4.1: Characteristics of popular windows.

Each window is characterized by the main-lobe width and the side-lobe level. The larger the main-lobe width the smaller is the decimation that I can introduce between the analysis and synthesis stages. This has a consequence in the choice of the hop size. For instance, using Hann[1] or Hamming windows I have to use at least a hop size equal to $R/2$ in order to preserve all information at the analysis stage. Moreover, the larger the main-lobe width, the more difficult is to separate two frequency components that are close to each other. In other words, we have a reduction in frequency resolution for windows with a large main lobe.

The side-lobe level indicates how much a sinusoidal component affects the DFT bins nearby. This phenomenon, called leakage, can induce an analysis procedure to detect false spectral peaks, or measurements on actual peaks can be affected by errors. For a given resolution considered to be acceptable, it is desirable that the side-lobe level be as small as possible.

The window length is chosen according to the tradeoff between spectral resolution and temporal resolution governed by the uncertainty principle. The

[1]The Hann window is often called Hanning window, probably for the same reason that in the US you may prefer saying "I xerox this document" rather than "I copy this document using a Xerox copier".

STFT analysis is based on the assumption that, within one frame, the signal is stationary. The more the window is short, the closer the assumption is to truth, but short windows determine low spectral resolution.

The windows described in this section have a fixed shape. When they are multiplied by an ideal lowpass impulse response they impose a fixed transition bandwidth, i.e. a certain frequency space between the passband and the stopband. There are other, more versatile windows, that allow to tune their behavior by means of a parameter. The most widely used of these adjustable windows is the Kaiser window [58], whose parameter β can be related to the transition bandwidth.

Zero padding

It is quite common to use a window whose length R is smaller than the number N of points used to compute the DFT. In this way, we have a spectrum representation on a larger number of points, and the shape of the frequency response can be understood more easily. Usually, the sequency of R points is extended by means of $N - R$ zeros, and this operation is called zero padding. Extending the time response with zeros corresponds to sampling the frequency response more densely, but it does not introduce any increase in frequency resolution. In fact, the resolution is only determined by the length and shape of the effective window, and additional zeros can not change it.

Consider the zero-padded signal

$$y(n) = \begin{cases} x(n) & n = 0, \ldots, R-1 \\ 0 & n = R, \ldots, N-1 \end{cases} . \tag{13}$$

The DFT is found as

$$\begin{aligned} Y(k) &= \sum_{n=0}^{N-1} y(n) e^{\frac{-j2\pi kn}{N}} = \sum_{n=0}^{R-1} y(n) e^{\frac{-j2\pi kn}{N}} = \\ &= \text{Resampling}_N(X, R) , \end{aligned} \tag{14}$$

where the notation $\text{Resampling}_N(X, R)$ indicates the resampling on N points of R points of the discrete-time signal X, obtained as $\text{DFT}(x) = X$.

Exercise

Draw the time-domain shape and the frequency response of each of the windows of table 4.1. Then, using a Rectangular, a Hann, and a Blackman

window, analyze the signal

$$x(n) = 0.8 \sin(2\pi f_1 n/F_s) + \sin(2\pi f_2 n/F_s) , \quad (15)$$

where $f_1 = 0.2F_s$ and $f_2 = 0.23F_s$, using $N = R = 64$. See the effects of halfing and doubling $N = R$, and observe the presence of leakage. Finally, repeat the exercise with $R = 32$, and $N = 64$ or $N = 128$.

4.1.4 Representations

One of the most useful visual representations of audio signals is the sonogram, also called spectrogram, that is a color- or grey-scale rendition of the magnitude of the STFT, on a 2D plane where time and frequency are the orthogonal axes.

Figure 6 shows the sonogram of the signal analyzed in exercise 4.1.3. Time is on the horizontal axis and frequency is on the vertical axis. Another useful visualization is the 3D plot, also called waterfall plot in sound analysis programs, when the analysis frames are presented one after the other from back to front. Figure 7 shows the 3D representation of the same signal analysis of figure 6.

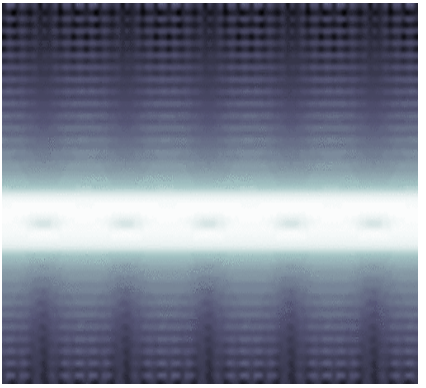

Figure 6: Sonogram representation of the signal (15). $N = 128$ and $R = 64$.

The Matlab signal processing toolbox, as well as the octave-forge project (see the appendix B), provide a function `specgram` that can be used to provide plots similar to those of figures 6 and 7. Specifically, these figures have been obtained by means of the octave script:

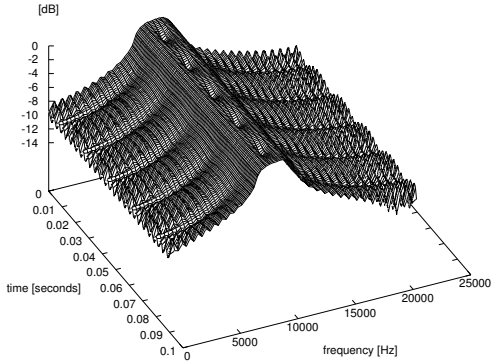

Figure 7: 3D STFT representation of the signal (15). $N = 128$ and $R = 64$.

```
Fs = 44100;
f1 = 0.2 * Fs;
f2 = 0.23 * Fs;
NMAX = 4096;
n = [1:NMAX];
x1 = 0.8 * sin (2*pi*f1/Fs*n);
x2 = sin (2*pi*f2/Fs*n);
y = x1 + x2;
N = 128;
R = 64;
[S,f,t] = specgram(y, N, Fs, hanning(R), R/2);
S = abs(S(2:N/2,:));        # magnitude in Nyquist range
S = S/max(S(:));            # normalize magnitude so
                            # that max is 0 dB.
imagesc(flipud(log(S)));    # display in log scale
mesh(t,f(1:length(f)-1),log(S));
gset view 35, 65, 1, 1.2
xlabel('time [seconds]');
ylabel('frequency [Hz]');
zlabel('[dB]');
replot;
```

In this example, the DFT length has been set to $N = 128$, the analysis

window is a Hann window with length $R = 64$, and the hop size to $R/2$. If the window length is doubled, the two components separate much more clearly, as shown in figure 8.

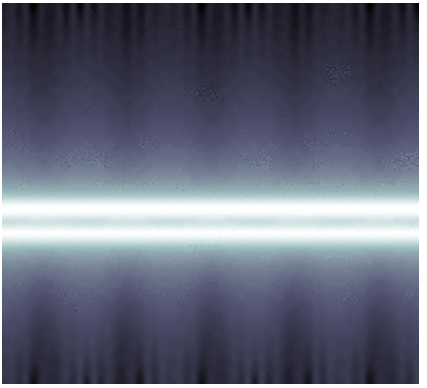

Figure 8: Sonogram representation of the signal (15). $N = 128$ and $R = 128$.

4.1.5 Accurate partial estimation

If the signal under analysis has a sinusoidal component that stays in between two adjacent DFT bins, the magnitude spectrum is similar to that reported in figure 9. We notice the two following phenomena:

- The sinusoidal component "leaks" some of its energy into bins that stay within a neighborhood of its theoretical position;
- It is difficult to determine the exact frequency of the component from visual inspection.

To overcome the latter problem, we describe two techniques: parabolic interpolation and phase following.

Parabolic interpolation

Any kind of interpolation can be applied to estimate the value and position of a frequency peak in the magnitude spectrum of a signal. Degree-two polynomial interpolation, i.e. parabolic interpolation, is particularly convenient as it uses only three bins of the magnitude spectrum.

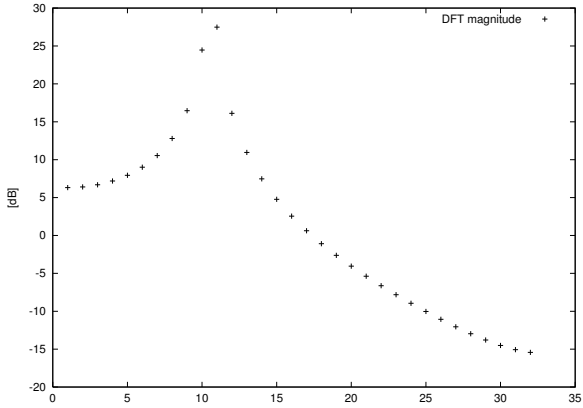

Figure 9: DFT image (magnitude) of a sinusoidal component.

Taken three adjacent bins of the magnitude DFT, we assign them the coordinates (x_0, y_0), (x_1, y_1), and (x_2, y_2). Then, we simply apply the Lagrange interpolation formula

$$\begin{aligned} y &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \\ &\quad + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2 \ . \end{aligned} \tag{16}$$

Since

$$x_1 - x_0 = x_2 - x_1 = \Delta f = \frac{F_s}{N} \tag{17}$$

is the frequency quantum, any point in the parabola has coordinates (x, y) related by

$$\begin{aligned} y &= [(x-x_1)(x-x_2)y_0 - 2(x-x_0)(x-x_2)y_1 + \\ &\quad + (x-x_0)(x-x_1)y_2] \, \frac{1}{2\Delta f^2} \ . \end{aligned} \tag{18}$$

From this expression, it is straightforward to find the peak as the point where the derivative vanishes: $y' = \frac{dy}{dx} = 0$.

Phase following

Let us assume that the signal to be analyzed can be expressed as a sum of sinusoids with time-varying amplitude and frequency (sinusoidal model, see sec. 5.1.1):

$$y(t) = \sum_{i=1}^{I} A_i(t) e^{j\phi_i(t)} \ , \tag{19}$$

with

$$\phi_i(t) = \int_{-\infty}^{t} \omega_i(\tau) d\tau \ , \tag{20}$$

being ω_i the frequency of the i-th partial.

For clarity, let us consider a signal containing only the i-th partial. The k-th bin of the m-th frame of the STFT gives

$$Y_m(k) = \sum_{n=0}^{N-1} w(m-n) A_i(n) e^{j\phi_i(n)} e^{-j\frac{2\pi}{N}kn} \tag{21}$$

$$= e^{-j\frac{2\pi}{N}km} \sum_{r=m-N+1}^{m} w(r) A_i(m-r) e^{j\phi_i(m-r)} e^{j\frac{2\pi}{N}kr} \ . \tag{22}$$

In order to proceed with the accurate partial frequency estimation, we have to make a

Assumption 1 Frequency and amplitude of the i-th component are constant within a STFT frame:

$$\phi_i(m-r) = \phi_i(m) - r\omega_i \tag{23}$$

$$A_i(m-r) = A_i(m) \ . \tag{24}$$

We see that

$$Y_m(k) = e^{-j\frac{2\pi}{N}km} A_i(m) e^{j\phi_i(m)} W(\frac{2\pi}{N}k - \omega_i) \ , \tag{25}$$

where $A_i(m)e^{j\phi_i(m)}$ contains the amplitude and instantaneous phase of the sinusoid that falls within the k-th bin, and $W(\frac{2\pi}{N}k - \omega_i)$ is the window transform. If we have access to the instantaneous phase, we can deduce the instantaneous frequency by back difference between two adjacent frames. This can be done as long as we deal with the problem of phase unwrapping, due to the fact that the phase is known modulo 2π.

It can be shown [52, pag. 287–288] that phase unwrapping can be unambiguous under

Assumption 2 Said H the hop size and $\frac{2\pi}{N}$ the separation between adjacent bins, let

$$\frac{2\pi}{N}H < \pi \ . \tag{26}$$

The assumption 2 holds for rectangular windows and imposes $H < \frac{N}{2}$. For Hann or Hamming windows the hop size must be such that $H < \frac{N}{4}$ (75% overlap). Therefore the frame rate to be used for accurate partial estimation is higher than the minimal frame rate needed for perfect reconstruction.

4.2 Linear predictive coding (with Federico Fontana)

The analysis/synthesis method known as linear predictive coding (LPC) was introduced in the sixties as an efficient and effective mean to achieve synthetic speech and speech signal communication [92]. The efficiency of the method is due to the speed of the analysis algorithm and to the low bandwidth required for the encoded signals. The effectiveness is related to the intelligibility of the decoded vocal signal.

The LPC implements a type of vocoder [10], which is an analysis/synthesis scheme where the spectrum of a source signal is weighted by the spectral components of the target signal that is being analyzed. The phase vocoder of figures 2 and 5 is a special kind of vocoder where amplitude and phase information of the analysis channels are retained and can be used as weights for complex sinusoids in the synthesis stage.

In the standard formulation of LPC, the source signals are either a white noise or a pulse train, thus resembling voiced or unvoiced excitations of the vocal tract, respectively.

The basic assumption behind LPC is the correlation between the n-th sample and the P previous samples of the target signal. Namely, the n-th signal sample is represented as a linear combination of the previous P samples, plus a residual representing the prediction error:

$$x(n) = -a_1 x(n-1) - a_2 x(n-2) - \cdots - a_P x(n-P) + e(n) \ . \tag{27}$$

Equation (27) is an autoregressive formulation of the target signal, and the analysis problem is equivalent to the identification of the coefficients $a_1, \ldots a_P$

of an allpole filter. If we try to minimize the error in a mean square sense, the problem translates into a set of P equations

$$\sum_{k=1}^{P} a_k \sum_n x(n-k)x(n-i) = -\sum_n x(n)x(n-i) \ , \tag{28}$$

or

$$\sum_{k=1}^{P} a_k R(i-k) = -R(i) \ , i = 1, \ldots, P \ , \tag{29}$$

where

$$R(i) \stackrel{\triangle}{=} \sum_n x(n)x(n-i) \tag{30}$$

is the signal autocorrelation.

In the z domain, equation (27) reduces to

$$E(z) = A(z)X(z) \tag{31}$$

where $A(z)$ is the polynomial with coefficients $a_1 \ldots a_P$. In the case of voice signal analysis, the filter $1/A(z)$ is called the allpole formant filter because, if the proper order P is chosen, its magnitude frequency response follows the envelope of the signal spectrum, with its broad resonances called formants. The filter $A(z)$ is called the inverse formant filter because it extracts from the voice signal a residual resembling the vocal tract excitation. $A(z)$ is also called a whitening filter because it produces a residual having a flat spectrum. However, we distinguish between two kinds of residuals, both having a flat spectrum: the pulse train and the white noise, the first being the idealized vocal-fold excitation for voiced speech, the second being the idealized excitation for unvoiced speech. In reality, the residual is neither one of the two idealized excitations. At the resynthesis stage the choice is either to use an encoded residual, possibly choosing from a code book of templates, or to choose one of the two idealized excitations according to a voiced/unvoiced decision made by the analysis stage.

When the target signal is periodic (voiced speech), a pitch detector can be added to the analysis stage, so that the resynthesis can be driven by periodic replicas of a basic pulse, with the correct inter-pulse period. Several techniques are available for pitch detection, either using the residual or the target signal [53]. Although not particularly efficient, one possibility is to do a Fourier analysis of the residual and estimate the fundamental frequency by the techniques of section 4.1.5.

Summarizing, the information extracted in a frame by the analysis stage are:

- the prediction coefficients $a_1, \ldots, a_P$;
- the residual e;
- pitch of the excitation residual;
- voiced/unvoiced information;
- signal energy (RMS amplitude).

These parameters, possibly modified, are used in the resynthesis, as explained in section 5.1.3.

The equations (29) are solved via the well-known Levinson-Durbin recursion [53], which provides the reflection coefficients of the lattice realization of the filter $1/A(z)$. As we mentioned in section 2.2.4, the reflection coefficients are related to a piecewise cylindrical modelization of the vocal tract. The LPC analysis proceeds by frames lasting a few milliseconds. In each frame the signal is assumed to be stationary and a new estimation of the coefficients is made. For the human vocal tract, $P = 12$ is a good estimate of the degrees of freedom that are needed to represent most articulations.

Besides its applications in voice coding and transformation, LPC can be useful whenever it is necessary to represent the shape of a stationary spectrum. Spectral envelope extraction by LPC analysis can be accurate as long as the filter order is carefully chosen, as depicted in figure 10. The accuracy depends on the kind of signal that is being analyzed, as the allpole nature of the LPC filter gives a spectral envelope with rather sharp peaks.

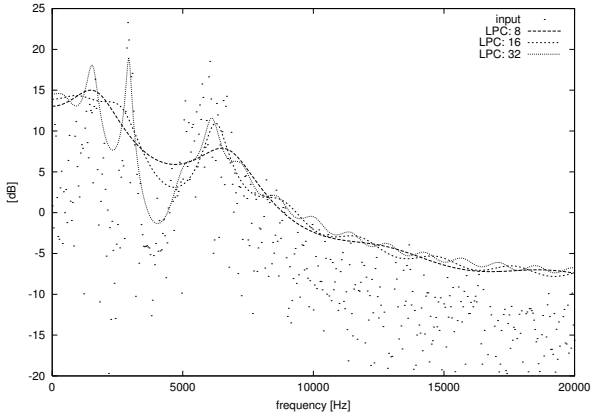

Figure 10: DFT image (magnitude) of a target signal and frequency response of allpole filters, identified via LPC with three different values of the order P.

Chapter 5

Sound Modelling

5.1 Spectral modelling

5.1.1 The sinusoidal model

A sound is expressed according to the sinusoidal model if it has the form

$$y(t) = \sum_{i=1}^{I} A_i(t) e^{j\phi_i(t)} \ , \tag{1}$$

where $\phi_i(t) = \int_{-\infty}^{t} \omega_i(\tau) d\tau$, and $A_i(t)$ and $\omega_i(t)$ are the i-th sinusoidal-component instantaneous magnitude and frequency, respectively. In practice, we consider discrete-time real signals. Therefore, we can write

$$y(n) = \sum_{i=1}^{I} A_i(n) \cos\left(\phi_i(n)\right) \ , \tag{2}$$

with

$$\phi_i(n) = \int_{0}^{nT} \omega_i(\tau) d\tau + \phi_{0,i} \ . \tag{3}$$

In principle, if I is arbitrarily high, any sound can be expressed according to the sinusoidal model. This principle states the generality of the additive synthesis approach. Actually, the noise components would require a multitude of sinusoids, and it is therefore convenient to treat them separately by introduction

of a "stochastic" part $e(n)$:

$$y(n) = \underbrace{\sum_{i=0}^{I} A_i(n)\cos\left(\phi_i(n)\right)}_{\text{Deterministic Part}} + \underbrace{e(n)}_{\text{Stochastic Part}} . \tag{4}$$

The separation of the stochastic part from the deterministic part can be done by means of the Short-Time Fourier Transform using the scheme of figure 1. Here, we rely on the fact that the STFT analysis retains the phases of the sinusoidal components, thus allowing a reconstruction that preserves the wave shape [94]. In this way, the deterministic part can be subtracted from the original signal to give the stochastic residual. One popular implementation of the scheme in figure 1 is found in the software `sms`, an acronym for spectral modeling synthesis[1] [5].

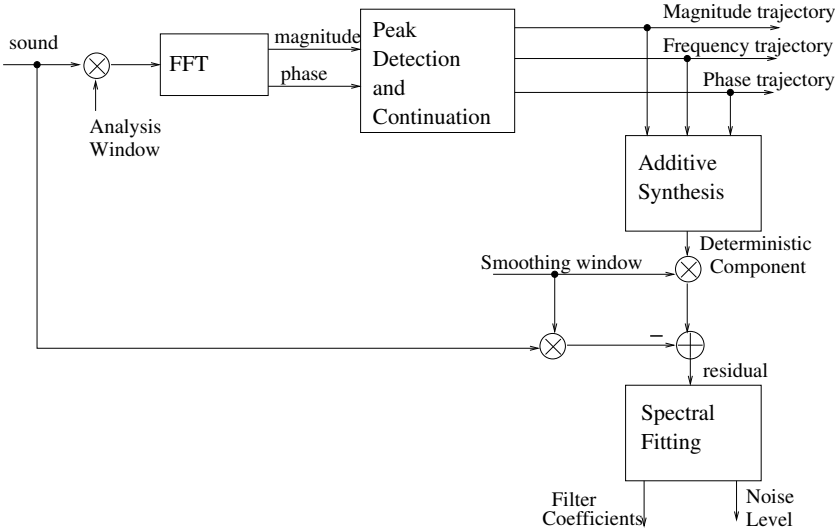

Figure 1: Separation of the sinusoidal components from a stochastic residual.

[1] The executable of `sms` is freely downloadable from http://www.iua.upf.es/~sms/

Peak detection and continuation

In order to separate the sinusoidal part from the residual we have to detect and track the most prominent frequency peaks, as they are indicators of strong sinusoidal components. One strategy is to draw "guides" across the STFT frames [94], in such a way that prolongation by continuity fills local holes that may occur in peak trajectories. If a guide detects missing evidence of the supporting peak for more than a certain number of frames, the guide is killed. Similarly, we start new guides as long as we detect a persistent peak. Therefore, the generation and destruction of peaks is governed by hysteresis (see figure 2).

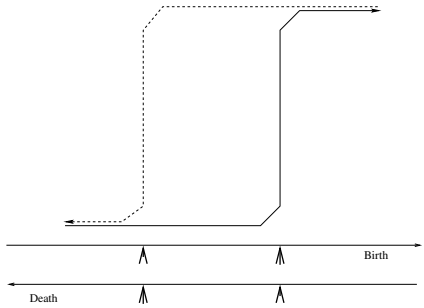

Figure 2: Hysteretic procedure for guide activation and destruction.

In order to better capture the deterministic structure during transients, it is better to run the analysis backward in time, since in most cases a sharp attack is followed by a stable release, and peak tracking is more effective when stable states are reached gradually and suddenly released, rather than vice versa.

If we can rely on the assumption of harmonicity of the analyzed sounds, the partial tracking algorithm can be "encouraged" by superposition of a harmonic comb onto the spectral profile.

For a good separation, frequencies and phases must be determined accurately, following the procedures described in section 4.1.5. Moreover, for the purpose of smooth resynthesis, the amplitudes of partials should be interpolated between frames, the most common choice being linear interpolation. Frequencies and phases should be interpolated as well, but one should be careful to ensure that the frequency track is always the derivative of the phase track. Since a third-order polynomial is uniquely determined by four degrees of freedom, by using a cubic interpolating polynomial one may impose the instantaneous

phases and frequencies between any couple of frames.

Resynthesis of the sinusoidal components

In the resynthesis stage, the sinusoidal components can be generated by any of the methods described in section 5.2, namely the digital oscillator in wavetable or recursive form, or the FFT-based technique. The latter will be more convenient when the sound has many sinusoidal components.

The DTFT of a windowed sinusoidal signal is the transform of the window, centered on the frequency of the sinusoid, and multiplied by a complex number whose magnitude and phase are the magnitude and phase of the sine wave. A signal that is the weighted sum of sinusoids gives rise, in the frequency domain, to a weighted sum of window transforms centered around different central frequencies.

If the window has a

A. sufficiently-high sidelobe attenuation,

we are allowed to consider only a restricted neighborhood of the window transform peak. The sound resynthesis can be achieved by anti-transformation of a series of STFT frames, and by the procedure of overlap and add applied to the time-domain frames. The signal reconstruction is free of artifacts if

B. the shifted copies of the window overlap and add to give a constant.

If w is the window that fulfills property (A), and Δ is the window that fulfills property (B), we can use w for the analysis and multiply the sequence by Δ/w after the inverse transformation [35]. Using two windows gives good flexibility in satisfying both the requirements (A) and (B). A particularly simple and effective window that satisfies property (B) is the triangular window.

This FFT-based synthesis (or FFT^{-1} synthesis) is convenient when the sinusoidal model gives many sine components, because its complexity is largely due to the cost of FFT, which is independent on the number of components. It is quite easy to introduce noise components with arbitrary frequency distribution just by adding complex numbers with the desired magnitude (and arbitrary phase) in the frequency domain.

Extraction of the residual

The extraction of a broad-spectrum noise residual could be performed either in the frequency domain or, as proposed in figure 1, directly by subtraction in

the time domain. This is possible because the STFT analysis preserves the information on phase, thus allowing a waveshape preservation. The stochastic component can be itself represented on a frame-by-frame basis, but the corresponding frame can be smaller than the analysis frame so that transients are captured more accurately.

Residual spectral fitting

The stochastic component is modeled as broad-band noise filtered by a linear coloring block. Such decomposition corresponds to a subtractive synthesis model [78], whose parameters may be obtained by LPC analysis (see section 4.2). However, if the purpose of the sines-plus-noise decomposition is that of sound modification, it is more convenient to model the stochastic part in the frequency domain. The magnitude spectrum of the residual can be approximated by means of a piecewise-linear function, that is described by the coordinates of the joints. The time-domain resynthesis can be operated in the time domain by inverse FFT, after having imposed the desired magnitude profile and a random phase profile.

Sound modifications

The sinusoidal model is interesting because it allows to apply musical transformations to sounds that are taken from actual recordings. The separation of the stochastic residual from the sinusoidal part allows a separate treatment of the two components.

Examples of musical transformations are:

Coloring: The spectral profile can be changed at will;

Emphasizing: The stochastic or the sinusoidal components can be exaggerated;

Time Stretching: the temporal extension of the sound can be altered without pitch modifications and with limited artifacts;

Pitch Shifting: The pitch can be transposed without changing the sound length and with limited artifacts;

Morphing: for instance,

- The spectral envelope of a sound can be imposed to another sound;

- A residual from a different sound can be used for resynthesis.

Figure 3 shows the framework for performing these musical modifications.

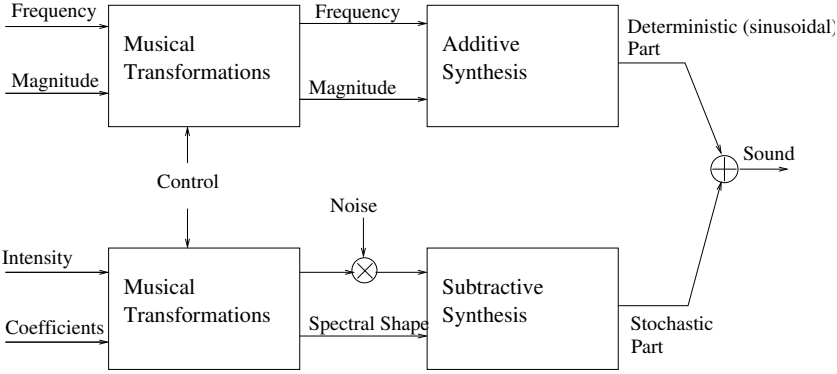

Figure 3: Framework for performing music transformations.

5.1.2 Sines + Noise + Transients

The fundamental assumption behind the sinusoids + noise model is that sound signals are composed of slowly-varying sinusoids and quasi-stationary broadband noises. This view is quite schematic, as it neglects the most interesting part of sound events: transients. Sound modifications would be much more easily achieved if transients could be taken apart and treated separately. For instance, in most musical instruments extending the duration of a note does not have any effect on the quality of the attack, which should be maintained unaltered in a time-stretching task.

For these reasons, a new sines + noise + transients (SNT) framework for sound analysis was established [108]. The key idea of practical transient extraction comes from the observation that, as sinusoidal signals in the time domain are mapped to well-localized spikes in the frequency domain, by duality short pulses in the time domain would correspond to sine-like curves in the frequency domain. Therefore, the sinusoidal model can be applied in the frequency domain to represent these sinusoidal components. The scheme of the SNT decomposition is represented in figure 4.

The DCT block in figure 4 represents the operation of Discrete Cosine

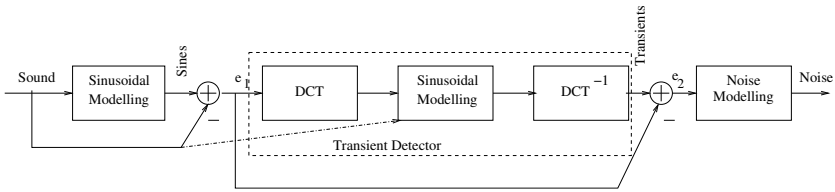

Figure 4: Decomposition of a sound into sines + noise + transients.

Transform, defined as

$$C(k) = \alpha \sum_{n=0}^{N-1} x(n) \cos\left(\frac{(2n+1)k\pi}{2N}\right). \tag{5}$$

The DCT has the property that an impulse is transformed into a cosine, and a cluster of impulses becomes a superposition of cosines. Therefore, in the transformed domain it makes sense to use the sinusoidal model and to extract a second residue that is given by transient components.

5.1.3 LPC Modelling

As explained in section 4.2, the Linear Predictive Coding can be used to model piecewise stationary spectra. The LPC synthesis proceeds according to the feedforward scheme of figure 5. Essentially, it is a subtractive synthesis algorithm where a spectrally-rich excitation signal is filtered by an allpole filter. The excitation signal can be the residual e that comes directly from the analysis, or it is selected from a code book. Alternatively, we can make use of voiced/unvoiced information to generate an excitation signal that can be either a random noise or a pulse train. In the latter case, the pulse repetition period is derived from pitch information, available as a parameter.

Between the analysis and synthesis stages, several modifications are possible:

- pitch shifting, obtained by modification of the pitch parameter;
- time stretching, obtained by stretching the window where the signal is assumed to be stationary;
- data reduction, by model order reduction or residual coding.

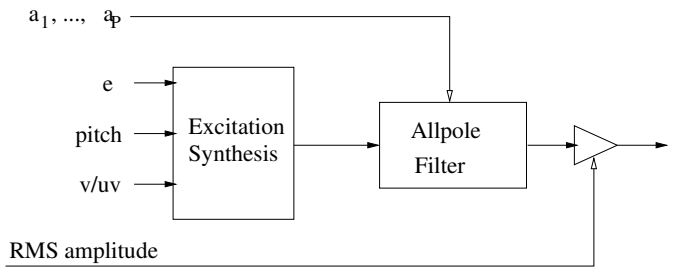

Figure 5: LPC Synthesis

5.2 Time-domain models

While the description of sound is more meaningful if done in the spectral domain, in many applications it is convenient to approach the sound synthesis directly in the time domain.

5.2.1 The Digital Oscillator

We have seen in section 5.1.1 how a complex sound made of several sinusoidal partials is conveniently synthesized by the FFT^{-1} method. If the sinusoidal components are not too many, it may be convenient to synthesize each partial by means of a digital oscillator.

From the obvious identity

$$e^{j\omega_0(n+1)} = e^{j\omega_0} e^{j\omega_0 n} , \tag{6}$$

said $e^{j\omega_0 n} = x_R(n) + jx_I(n)$, it is evident that the oscillator can be implemented by one complex multiplication, i.e., 4 real multiplications, at each time step:

$$x_R(n+1) = \cos\omega_0 x_R(n) - \sin\omega_0 x_I(n) \tag{7}$$

$$x_I(n+1) = \sin\omega_0 x_R(n) + \cos\omega_0 x_I(n) . \tag{8}$$

The initial amplitude and phase can be imposed by scaling the initial phasor $e^{j\omega_0 0}$ and adding a phase shift to its exponent. It is easy to show[2] that the calculation of $x_R(n+1)$ can also be performed as

$$x_R(n+1) = 2\cos\omega_0 x_R(n) - x_R(n-1) , \tag{9}$$

[2]The reader is invited to derive the difference equation 9

or, in other words, as the free response of the filter

$$H_R(z) = \frac{1}{1 - 2\cos\omega_0 z^{-1} + z^{-2}} = \frac{1}{(1 - e^{-j\omega_0} z^{-1})(1 - e^{j\omega_0} z^{-1})} \ . \quad (10)$$

The poles of the filter (10) lay exactly on the unit circumference, at the limit of the stability region. Therefore, after the filter has received an initial excitation, it keeps ringing forever.

If we call x_{R1} and x_{R2} the two state variables containing the previous samples of the output variable x_R, an initial phase ϕ_0 can be imposed by setting[3]

$$x_{R1} = \sin(\phi_0 - \omega_0) \quad (11)$$
$$x_{R2} = \sin(\phi_0 - 2\omega_0) \ . \quad (12)$$

The digital oscillator is particularly convenient to perform sound synthesis on general-purpose processors, where floating-point arithmetics is available at no additional cost. However, this method for generating sinusoids has two main drawbacks:

- Updating the parameter (i.e., the oscillation frequency) requires computing a cosine function. This is a problem for audio rate modulations, where to compute a modulated sine we need to compute a cosine at each time sample.
- Changing the oscillation frequency changes the sinusoid amplitude as well. Therefore, some amplitude control logic is needed.

5.2.2 The Wavetable Oscillator

The most classic and versatile approach to the synthesis of periodic waveforms (sinusoids included) is the cyclic reading of a table where a waveform period is pre-stored. If the waveform to be synthesized is a sinusoid, symmetry considerations allow to store only one fourth of the period and play with the index arithmetic to reconstruct the whole period.

Call `buf[]` the buffer that contains the waveform period, or wavetable. The wavetable oscillator works by circularly accessing the wavetable at multiples of an increment I and reading the wavetable content at that position.

[3]The reader can verify, using formulas (29–32) of appendix A, that $x_R(0) = \sin\phi_0$, given $x_R(-1) = x_{R1}$ and $x_R(-2) = x_{R2}$.

If B is the buffer length, and f_0 is the frequency that we want to generate at the sample rate F_s, the increment has to be set to

$$I = \frac{B f_0}{F_s} \ . \tag{13}$$

It is easy to realize that the reading pointer accesses the wavetable at indexes that are, in general, fractional. Therefore, some form of interpolation has to be used. The following strategies have an increasing degree of accuracy (and complexity):

Truncation: `buf[`$\lfloor$`index`$\rfloor$`]`

Rounding: `buf[`$\lfloor$`index` $+ 0.5 \rfloor$`]`

Linear Interpolation: `buf[`$\lceil$`index`$\rceil$`]` $($`index` $-$ $\lfloor$`index`$\rfloor)$ +
`buf[`$\lfloor$`index`$\rfloor$`]` $(1 -$ `index` $+$ $\lfloor$`index`$\rfloor)$

Higher-order polynomial interpolation

"Multirate" interpolation: the problem is re-casted as a sampling-rate conversion.

By increasing the complexity of interpolation it is possible, given a certain level of acceptable digital noise, to decrease the wavetable size [41]. The linear interpolation is particularly attractive for implementations in custom or specialized hardware (see section B.5.1 of the appendix B). The most-significant bits of the index can be used to access the buffer locations, and the least-significant bits are used to approximate the quantity $($`index` $-$ $\lfloor$`index`$\rfloor)$ in the computation of the interpolation.

Sampling-rate conversion

The problem of designing a wavetable oscillator can be re-casted as a problem of sampling-rate conversion, i.e., transforming a signal sampled at rate $F_{s,1}$ into its copy re-sampled at rate $F_{s,2}$. If $\frac{F_{s,2}}{F_{s,1}} = \frac{L}{M}$, with L and M irreducible integers, we can re-sample by:

1. Up-sampling by a factor L
2. Low-pass filtering
3. Down-sampling by a factor M .

Figure 6 represents these three operations as a cascade of linear (but non-time-invariant) blocks, where the upward arrow denots upsampling (or introducing zeros between non-zero samples) and the downward arrow denotes downsampling (or decimating).

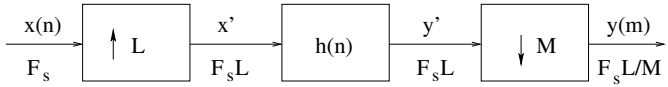

Figure 6: Block decomposition of re-sampling

Figure 7 shows the spectral effects of the various stages of resampling when $L/M = 3/2$.

If the interpolation is realized by sampling-rate conversion the problem reduces to designing a good lowpass filter. However, since the resampling ratio L/M changes for each different pitch that is obtained from the same wavetable, the characteristics of the lowpass filter have to be made pitch-dependent. Alternatively, a set of filters can be designed to accomodate all possible pitches, and the appropriate coefficient set is selected at run time [55].

5.2.3 Wavetable sampling synthesis

The wavetable sampling synthesis is the extension of the wavetable oscillator to

- Non-sinusoidal waveforms;
- Wavetables storing several periods.

Usually, this kind of sound synthesis is based on the following tricks:

- The attack transient is reproduced "faithfully" by straight sampling;
- A selection of periods of the central part of the sound (sustain) is stored in a buffer and cyclically read (loop). The increment is selected in order to produce the desired pitch;
- The keyboard[4] is divided into segments of contiguous notes (splits). Each split uses transpositions of the same sample;

[4]The keyboard metaphor is used very often even for sound timbres that do not come from keyboard instruments.

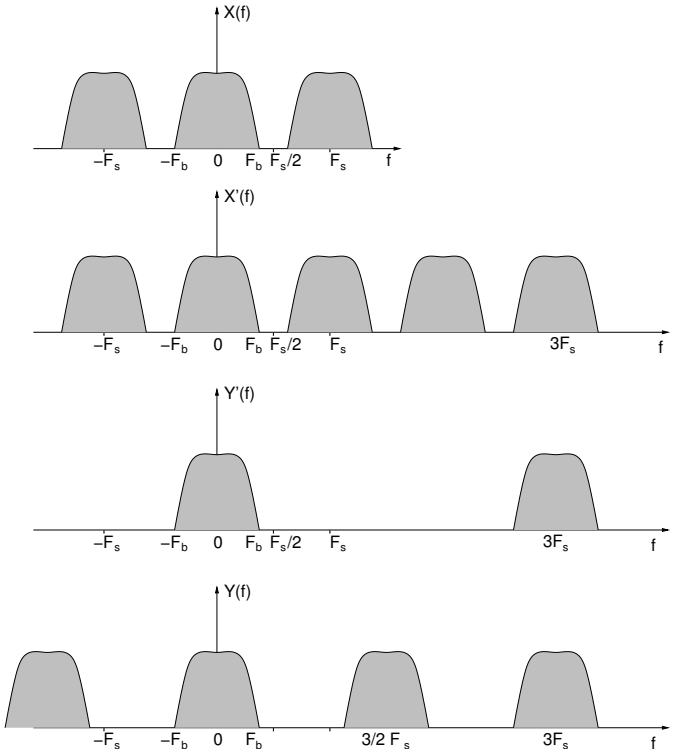

Figure 7: Example of re-sampling with $L/M = 3/2$

- Different dynamic levels are obtained by
  - Sampling at different dynamic levels and obtaining the intermediate samples by interpolation, or
  - Sampling *fortissimo* notes and obtaining lower intensities by dynamic filtering (usually lowpass).

In wavetable sampling synthesis, the control signals are extremely important to achieve a natural sound behavior. The control signals are tied to the evolution of the musical gesture, thus evolving much more slowly than audio signals. Therefore, a control rate can be used to generate signals for

- Temporal envelopes (e.g., Attack - Decay - Sustain - Release);

- Low-Frequency Oscillators (LFO) for vibrato and tremolo;
- Dynamic control of filters.

5.2.4 Granular synthesis (with Giovanni De Poli)

Short wavetables can be read at different speeds and the resulting sound grains can be concatenated and overlapped in time. This time-domain approach to sound synthesis is called granular synthesis. Granular synthesis starts from the idea of analyzing sounds in the time domain by representing them as sequences of short elements called "grains". The parameters of this technique are the waveform of the grain $g_k(\cdot)$, its temporal location l_k and amplitude a_k

$$s_g(n) = \sum_k a_k g_k(n - l_k) \ . \tag{14}$$

A complex and dynamic acoustic event can be constructed starting from a large quantity of grains. The features of the grains and their temporal locations determine the sound timbre. We can see it as being similar to cinema, where a rapid sequence of static images gives the impression of objects in movement. The initial idea of granular synthesis dates back to Gabor [26], while in music it arises from early experiences of tape electronic music. The choice of parameters can be via various criteria driven by interpretation models. In general, granular synthesis is not a single synthesis model but a way of realizing many different models using waveforms that are locally defined. The choice of the interpretation model implies operational processes that may affect the sonic material in various ways.

The most important and classic type of granular synthesis (asynchronous granular synthesis) distributes grains irregularly on the time-frequency plane in form of clouds [77]. The grain waveform is

$$g_k(i) = w_d(i) \cos(2\pi f_k T_s i) \ , \tag{15}$$

where $w_d(i)$ is a window of length d samples, that controls the time span and the spectral bandwidth around f_k. For example, randomly scattered grains within a mask, which delimits a particular frequency/amplitude/time region, result in a sound cloud or musical texture that varies over time. The density of the grains within the mask can be controlled. As a result, articulated sounds can be modeled and, wherever there is no interest in controlling the microstructure exactly, problems involving the detailed control of the temporal characteristics

of the grains can be avoided. Another peculiarity of granular synthesis is that it eases the design of sound events as parts of a larger temporal architecture. For composers, this means a unification of compositional metaphors on different scales and, as a consequence, the control over a time continuum ranging from the milliseconds to the tens of seconds. There are psychoacoustic effects that can be easily experimented by using this algorithm, for example crumbling effects and waveform fusions, which have the corresponding counterpart in the effects of separation and fusion of tones.

5.3 Nonlinear models

5.3.1 Frequency and phase modulation

The most popular non-linear synthesis technique is certainly frequency modulation (FM). In electrical communications, FM has been used for decades, but its use as a sound synthesis algorithm in the discrete-time domain is due to John Chowning [23]. Essentially, Chowning was doing experiments on different extents of vibrato applied to simple oscillators, when he realized that fast vibrato rates produce dramatic timbral changes. Therefore, modulating the frequency of an oscillator was enough to obtain complex audio spectra.

Chowning's FM model is:

$$x(n) = A \sin\left(\omega_c n + I \sin\left(\omega_m n\right)\right) = A \sin\left(\omega_c n + \phi(n)\right) , \qquad (16)$$

where ω_c is called the carrier frequency, ω_m is called the modulation frequency, and I is the modulation index. Strictly speaking, equation (16) represents a phase modulation because it is the instantaneous phase that is driven by the modulator. However, when both the modulator and the carrier are sinusoidal, there is no substantial difference between phase modulation and frequency modulation. The instantaneous frequency of (16) is

$$\omega(n) = \omega_c - I\omega_m \cos\left(\omega_m n\right) , \qquad (17)$$

or, in Hertz,

$$f(n) = f_c - I f_m \cos\left(2\pi f_m n\right) . \qquad (18)$$

Figure 8 shows a `pd` patch implementing the simple FM algorithm. The modulation frequency is used to control an oscillator directly, while the carrier frequency controls a `phasor~` unit generator. This block generates the cyclical phase ramp that, when given as index of a cosinusoidal table, produces

the same result as the `osc` unit generator. However, this decomposition of the oscillator into two parts (i.e., the phase generation and the table read) allows to sum the output coming from the modulator directly to the phase of the carrier.

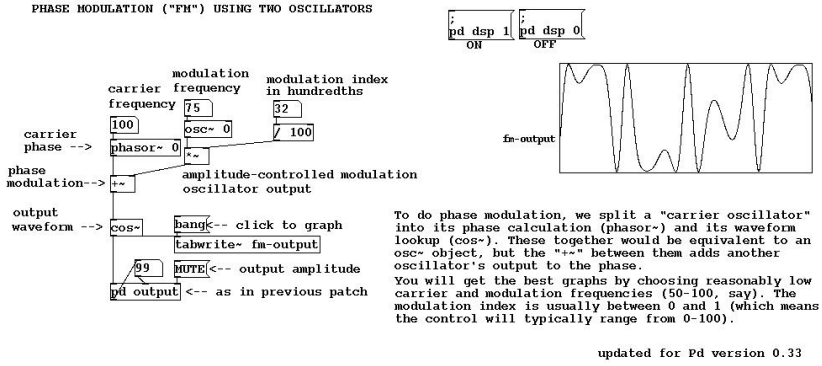

Figure 8: `pd` patch for phase modulation. Adapted from a help patch of the `pd` distribution.

Given the carrier and modulation frequencies, and the modulation index, it is possible to predict the distribution of components in the frequency spectrum of the resulting sound. This analysis is based on the trigonometric identity [1]

$$\begin{aligned} x(n) &= A\sin\left(\omega_c n + I\sin\left(\omega_m n\right)\right) \\ &= A\left\{\underbrace{J_0(I)\sin\left(\omega_c n\right)}_{\text{carrier}} + \right. \\ &\quad \left.\underbrace{\sum_{k=1}^{\infty} J_k(I)\left[\sin\left((\omega_c + k\omega_m)n\right) + (-1)^k\sin\left((\omega_c - k\omega_m)n\right)\right]}_{\text{side frequencies}}\right\}, \end{aligned} \tag{19}$$

where $J_k(I)$ is the k-th order Bessel function of the first kind. These Bessel functions are plotted in figure 9 for several values of k (number of side frequency) and I (modulation index).

Therefore, the effect of phase modulation is to introduce side components that are shifted in frequency from the fundamental by multiples of ω_m

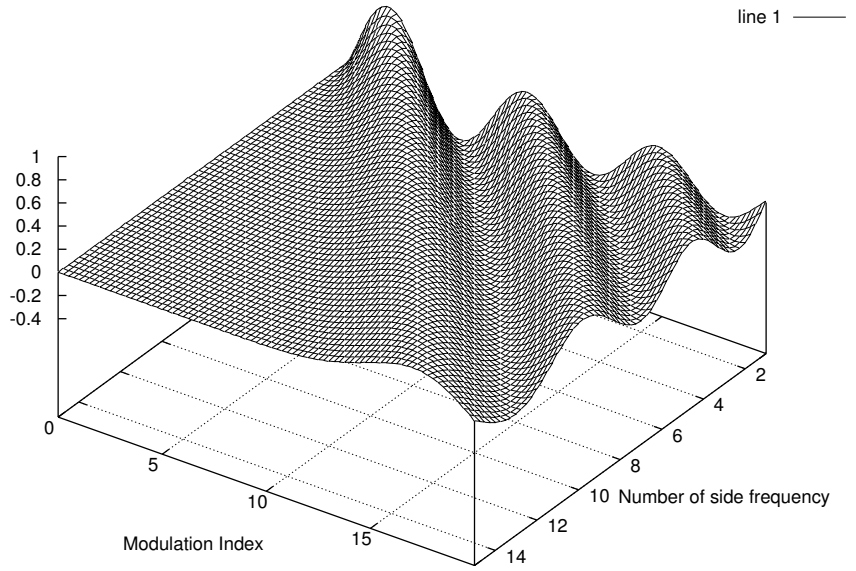

Figure 9: Bessel functions of the first kind

and whose amplitude is governed by $J_k(I)$. Generally speaking, the larger the modulation index, the wider is the sound bandwidth. Since the number of side components that are stronger than one hundredth of the carrier magnitude is approximately

$$M = I + 0.24I^{0.27} \ , \tag{20}$$

the bandwidth is approximately

$$\text{BW} = 2\left(I + 0.24I^{0.27}\right)\omega_m \approx 2I\omega_m \ . \tag{21}$$

If the ratio ω_c/ω_m is rational the resulting spectrum is harmonic, and the partials are multiple of the fundamental frequency

$$\omega_0 = \frac{\omega_c}{N_1} = \frac{\omega_m}{N_2} \ , \tag{22}$$

where

$$\frac{N_1}{N_2} = \frac{\omega_c}{\omega_m} \ , \text{with } N_1, N_2 \text{ irreducible couple} \ . \tag{23}$$

For instance, if $N_2 = 1$, all the harmonics are present, and if $N_2 = 2$ only the odd harmonics are present.

When calculating the spectral components, some of the partials on the left of the carrier may assume a negative frequency. Since $\sin(-\theta) = -\sin\theta = \sin(\theta - \pi)$, these components have to be flipped onto the positive axis and summed (magnitude and phase) with the components possibly already present at those frequencies.

Complex carrier

We can have a bank of oscillators sharing a single modulator or, equivalently, a non-sinusoidal carrier. In this case, each sinusoidal component of the complex carrier is enriched by side components as if it were the carrier of a simple FM couple.

One application of FM with a complex carrier is the construction of vowel-like spectra, as it was demonstrated by Chowning in the eighties. Each partial of the carrier may be associated with the center of one formant, i.e. a prominent lobe in the envelope of the magnitude spectrum. For a given person's voice, each vowel is characterised by a certain frequency distribution of formants.

Exercise

The reader is invited to implement an FM instrument (in, e.g., Octave or `pd`) that reproduces the vowel `/a/`, whose formants are found at 700, 1200, and 2500 Hz. How can a vibrato be implemented in such a way that the formant position remains fixed?

Complex modulator

The modulating waveform can be non-sinusoidal. In this case the analysis can be quite complicated. For instance, a modulator with two partials ω_1 and ω_2, acting on a sinusoidal carrier, gives rise to the expansion

$$x(n) = A \sum_k \sum_m J_k(I_1) J_m(I_2) \sin\left((\omega_c + k\omega_1 + m\omega_2)\, n\right) . \qquad (24)$$

Partials are found at the positions $|\omega_c \pm k\omega_1 \pm m\omega_2|$. If $\omega_M = \mathrm{MCD}(\omega_1, \omega_2)$, the spectrum has partials at $|\omega_c \pm k\omega_M|$. For instance, a carrier $f_c = 700$Hz and a modulator with partials at $f_1 = 200$Hz and $f_1 = 300$Hz, produce a harmonic

spectrum with fundamental at 100Hz. The advantage of using complex modulators in this case is that the spectral envelope can be controlled with more degrees of freedom.

Feedback FM

A sinusoidal oscillator can be used to phase-modulate itself. This is a feedback mechanism that, with a unit-sample feedback delay, can be expressed as

$$x(n) = \sin\left(\omega_c n + \beta x(n-1)\right) , \tag{25}$$

where β is the feedback modulation index. The trigonometric expansion

$$x(n) = \sum_k \frac{2}{k\beta} J_k(k\beta) \sin\left(k\omega_c n\right) \tag{26}$$

holds for the output signal. By a gradual increase of β we can gradually transform a pure sinusoidal tone into a sawtooth wave [78]. If the feedback delay is longer than one sample we can easily produce routes to chaotic behaviors as β is increased [12, 15].

FM with Amplitude Modulation

By introducing a certain degree of amplitude modulation we can achieve a more compact distribution of partials around the modulating frequency. In particular, we can use the expansion[5] [74]

$$\begin{aligned} e^{I\cos\left(\omega_m n\right)} \sin\left(\omega_c n + I \sin\left(\omega_m n\right)\right) \quad &= \quad \sin\left(\omega_c n\right) + \\ &+ \sum_{k=1}^{\infty} \frac{I^k}{k} \sin\left(\left(\omega_c + k\omega_m\right) n\right) , \end{aligned} \tag{27}$$

to produce a sequence of partials that fade out as $1/k$ in frequency, starting from the carrier. Figure 10 shows the magnitude spectrum of the sound produced by the mixed amplitude/frequency modulation (28) with carrier frequency at 3000Hz, modulator at 1500Hz, modulation index $I = 0.2$, and sample rate $F_s = 22100$Hz.

[5]The reader is invited to verify the expansion (28) using an `octave` script with `wm = 100; wc = 200; I = 0.2; n = [1:4096]; y1 = exp(I*cos(wm*n)) .* sin(wc*n + I*sin(wm*n));`

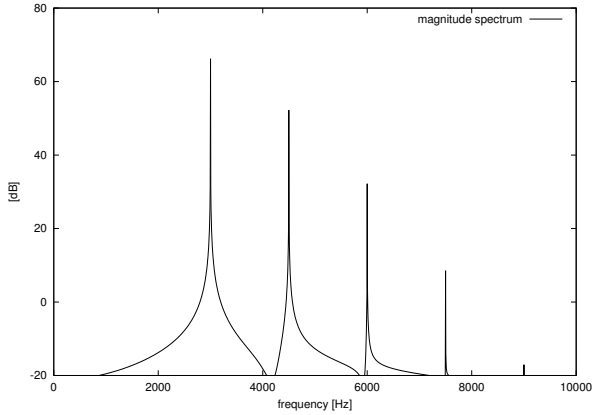

Figure 10: Spectrum of a sound produced by amplitude/frequency modulation as in (28).

Discussion

The synthesis by frequency modulation was very popular in the eighties, especially because it was implemented in the most successful synthesizer of all times: the Yamaha DX7. At that time, obtaining complex time-varying spectra with a few multiplies and adds was a major achievement. There was a theory that allowed to predict the spectra given the parameter, and the bandwidth of FM sounds could be controlled smoothly by means of the modulation index. However, it proved difficult to obtain FM patches starting from the analysis of real sounds, so that the most successful reproductions have been based on intuition and multiple trials. Some of the parameters, such as the carrier/modulator frequency ratio) are too critical and non-intuitive. Namely, little changes in a modulator frequency produce dramatic changes in timbre. The modulation index itself, despite displaying a global intuitive behavior, is related to each single partial amplitude by means of exotic functions that have no relationship with the human hearing system.

5.3.2 Nonlinear distortion

The sound synthesis by nonlinear distortion (NLD), or waveshaping [8], is conceptually very simple: the oscillator output is used as argument of a nonlinear function. In the discrete-time digital domain, the nonlinear function is

stored in a table, and the oscillator output is used as index to access the table.

The interesting thing about NLD is that there is a theory that allows to design the distorting table given certain specifications of the desired spectrum. If the oscillator is sinusoidal, we can formulate NLD as

$$x(n) = A\cos(\omega_0 n) \tag{28}$$
$$y(n) = F(x(n)) \; . \tag{29}$$

For the nonlinear function, we use Chebyshev polynomials [1]. The degree-n Chebyshev polynomial is defined by the recursive relation:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_n(x) &= 2xT_{n-1}(x) - T_{n-2}(x) \; , \end{aligned} \tag{30}$$

and it has the property

$$T_n(\cos\theta) = \cos n\theta \; . \tag{31}$$

In virtue of property (31), if the nonlinear distorting function is a degree-m Chebyshev polynomial, the output y, obtained by using a sinusoidal oscillator $x(n) = \cos\omega_0 n$, is $y(n) = \cos(m\omega_0 n)$, i.e., the m-th harmonic of x.

In order to produce the spectrum

$$y(n) = \sum_k h_k \cos(k\omega_0 n) \; , \tag{32}$$

it is sufficient to use the linear composition of Chebyshev functions

$$F(x) = \sum_k h_k T_k(x) \tag{33}$$

as a nonlinear distorting function.

Varying the oscillator amplitude A, the amount of distortion and the spectrum of the output sound are varied as well. However, the overall output amplitude does also vary as a side effect, and some form of compensation has to be introduced if a constant amplitude is desired. This is a clear drawback of NLD as compared to FM. Time-varying spectral variations can also be introduced by adding a control signal to the oscillator output x, so that the nonlinear function is dynamically shifted.

5.4 Physical models

Instead of trying to model the air pressure signal as it appears at the entrance of the ear canal, we can simulate the physical behavior of mechanical systems that produce sound as a side effect. If the simulation is accurate enough, we would obtain veridical sound dynamics and a detailed control in terms of physical variables. This allows direct manipulation of the sound synthesis model and direct coupling with gestural controllers.

5.4.1 A physical oscillator

Let us consider a simple mechanical mass-spring-damper system, as depicted in figure 11. Let f be an exogenous force that drives the system. It is a mechanical series connection, as the components share the same x position and the forces sum up to zero:

$$f_m = f_R + f_k + f \Rightarrow m\ddot{x} = -R\dot{x} - kx + f \ . \tag{34}$$

By taking the Laplace transform of (34) (with null initial conditions) we get

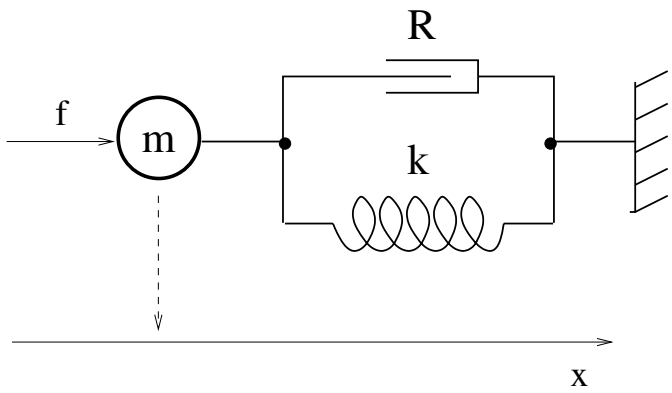

Figure 11: Mass-Spring-Damper system

the algebraic relationship

$$s^2 mX(s) + sRX(s) + kX(s) = F(s) \ , \tag{35}$$

and we can derive the transfer function between the forcing term f and the displacement x:

$$H(s) = \frac{X(s)}{F(s)} = \frac{1/m}{s^2 + \frac{R}{m}s + \frac{k}{m}} \; . \tag{36}$$

The system oscillates with characteristic frequency $\Omega_0 \stackrel{\triangle}{=} \sqrt{k/m} = 2\pi f_0$ and the damping coefficient is $\rho \stackrel{\triangle}{=} R/m$. The quality factor of the system is $Q = \Omega_0/\rho$ and it is the number of cycles that the characteristic oscillation takes to attenuate by a factor $1/e^{\pi}$. The damping coefficient ρ is proportional to the resonance bandwidth. If we use the bilinear transformation to discretize the transfer function (36) we obtain the discrete-time system described by the transfer function

$$\begin{aligned} H(z) &= \frac{1 + 2z^{-1} + z^{-2}}{mh^2 + Rh + k + 2(k - mh^2)z^{-1} + (k + mh^2 - Rh)z^{-2}} \\ &= \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} \end{aligned} \tag{37}$$

Therefore, the damped mechanical oscillator can be simulated by means of a second-order discrete-time filter. For instance, the realization Direct Form I, depicted in figure 24 of chapter 2, can be used for this purpose. We notice that there is a delay-free path that connects the input f with the output x, and this may represent a problem when connecting several simulations of physical blocks together.

5.4.2 Coupled oscillators

Let us consider the system obtained by coupling the mass-spring-damper oscillator with a second mass-spring system (see figure 12):

$$\begin{aligned} m_1\ddot{x}_1 &= -k_1(x_1 - x_2) - R(\dot{x}_1 - \dot{x}_2) + f \\ m_2\ddot{x}_2 &= -k_1(x_1 - x_2) - k_2 x_2 + R(\dot{x}_1 - \dot{x}_2) \; . \end{aligned} \tag{38}$$

Using the Laplace transform, the system (39) can be converted into

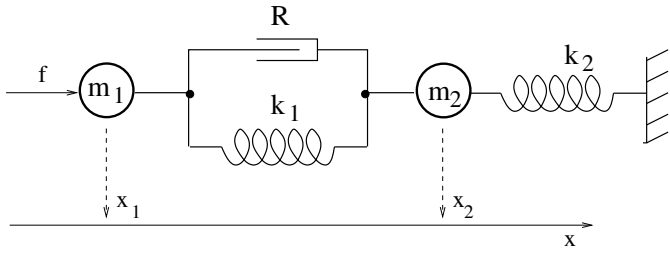

Figure 12: Two coupled mechanical oscillators

$$
\begin{aligned}
X_1(s) &= \frac{1}{m_1 s^2 + Rs + k_1}\left[F(s) + (k_1 + Rs)X_2(s)\right] \\
&\triangleq H_1(s)\left[F(s) + G(s)X_2(s)\right] \\
X_2(s) &= \frac{1}{m_2 s^2 + Rs + (k_1 + k_2)}(k_1 + Rs)X_1(s) \\
&\triangleq H_2(s)G(s)X_1(s)\ , \qquad (39)
\end{aligned}
$$

and this can be represented as a feedback connection of filters, as depicted in figure 13. This simple example gives us the possibility to discuss a few different

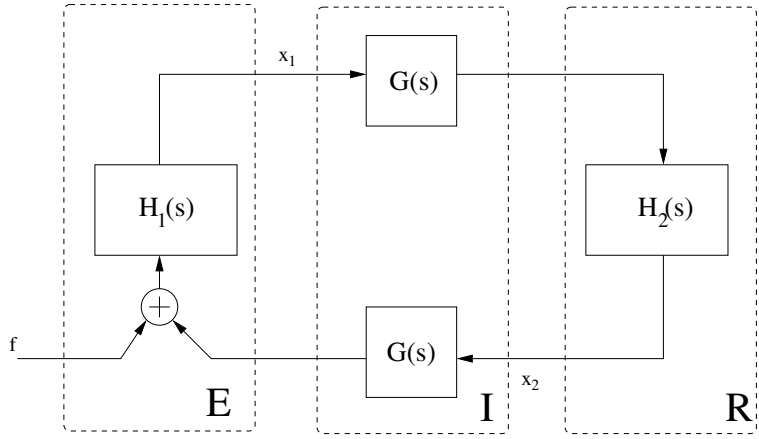

Figure 13: Block decomposition of the coupled oscillators

ways of looking at physical models. One of these ways is the cellular approach,

where complex linear systems are obtained by connection of mass points (H_1 and H_2 in our example) and visco-elastic links. Such approach is the basis of the CORDIS-ANIMA software developed at ACROE in Grenoble [20]. Another possibility, is to look for functional blocks in the system decomposition. In figure 13 we have outlined three functional blocks:

E - exciter: a dynamic physical system that can elicit and sustain an oscillation by means of an external forcing term;

R - resonator: a dynamic physical system (with small losses) that sustains the oscillations;

I - interaction: a system that connects E and R in such a way that the physical variables at the two ends are compatible.

Although in our example the resonator is a lumped mechanical oscillator, usually the resonator is a medium where waves propagate. Therefore the resonator is a distributed system, described by partial differential equations (PDE). Among the different ways of discretizing it, we mention

- Network of elementary coupled oscillators (cellular models);
- Numerical integration of the PDE (for instance, finite difference methods);
- Discretization of the solutions of the PDE (waveguide models).

The exciter is usually a lumped system described by ordinary differential equations (ODE) that can be integrated using numerical methods, the bilinear transformation, or the impulse invariance method. Often the exciter exhibits strong nonlinearities, such as the pressure-flow characteristic of a clarinet reed [31].

The interaction block is the place where the different discretizations of the exciter and resonator blocks talk to each other. Moreover, this is the right place to insert sound component that are difficult to capture with a physical model, either because the physics is too complicated or because we just don't know to model some phenomena. For instance, where the clarinet reed (exciter) is connected to the bore (resonator), small flow-dependent noise bursts can be injected to increase the simulation realism.

In a system such as the one of figure 13, if each block is separately discretized a computability problem may arise when the blocks are connected to each other. Namely, if the realization of each block has a delay-free input-output

path then a non-computable delay-free loop will appear in the model. There are techniques to cope with these delay-free loops (implicit solvers) or to eliminate them [16].

5.4.3 One-dimensional distributed resonators

Physical systems such as strings or acoustic tubes can be idealized as one-dimensional distributed resonators, described by a couple of dual variables, here called Kirchhoff variables, which are functions of time and longitudinal space. For a string, the Kirchhoff variables are force and velocity. For the acoustic tube, these variables are pressure and air flow. In any case, each of these variables is governed by the wave equation [63]

$$\frac{\partial^2 p(x,t)}{\partial t^2} = c^2 \frac{\partial^2 p(x,t)}{\partial x^2} \ , \tag{40}$$

where c is the wave speed in the medium. The symbol p in (40) can be thought of as the instantaneous and local air pressure inside a tube.

One of the most popular ways of solving PDEs such as (40) is finite differencing, where a grid is constructed in the spatial and time variables, and derivatives are replaced by linear combinations of the values on this grid. Two are the main problems to be faced when designing a finite-difference scheme for a partial differential equation: numerical losses and numerical dispersion. There is a standard technique [70], [103] for evaluating the performance of a finite-difference scheme in contrasting these problems: the von Neumann analysis. Replacing the second derivatives by central second-order differences[6], the explicit updating scheme for the i-th spatial sample of displacement (or pressure) is:

$$\begin{aligned} p(i,n+1) &= 2\left(1 - \frac{c^2\Delta t^2}{\Delta x^2}\right) p(i,n) - p(i,n-1) + \\ &\quad + \frac{c^2\Delta t^2}{\Delta x^2}\left[p(i+1,n) + p(i-1,n)\right] \ , \end{aligned} \tag{41}$$

where Δt and Δx are the time and space grid steps. The von Neumann analysis assumes that the equation parameters are locally constant and checks the time

[6]The reader is invited to derive (41) by substituting in (40) the first-order spatial derivative with the difference $(p(i+1,n) - p(i,n))/X$, and the first-order time derivative with the difference $(p(i,n+1) - p(i,n))/T$

evolution of a spatial Fourier transform of (41). In this way a spectral amplification factor is found whose deviations from unit magnitude and linear phase give respectively the numerical loss (or amplification) and dispersion errors. For the scheme (41) it can be shown that a unit-magnitude amplification factor is ensured as long as the Courant-Friedrichs-Lewy condition [70]

$$\frac{c\Delta t}{\Delta x} \leq 1 \tag{42}$$

is satisfied, and that no numerical dispersion is found if equality applies in (42). A first consequence of (42) is that only strings having length which is an integer number of $c\Delta t$ are exactly simulated. Moreover, when the string deviates from ideality and higher spatial derivatives appear (physical dispersion), the simulation becomes always approximate. In these cases, the resort to implicit schemes can allow the tuning of the discrete algorithm to the amount of physical dispersion, in such a way that as many partials as possible are reproduced in the band of interest [22].

It is worth noting that if c in equation (40) is a function of time and space, the finite difference method retains its validity because it is based on a local (in time and space) discretization of the wave equation. Another advantage of finite differencing over other modeling techniques is that the medium is accessible at all the points of the time-space grid, thus maximizing the possibilities of interaction with other objects.

As opposed to finite differencing, which discretize the wave equation (see eqs. (40) and (41)), waveguide models come from discretization of the solution of the wave equation. The solution to the one-dimensional wave equation (40) was found by D'Alembert in 1747 in terms of traveling waves [7]:

$$p(x,t) = p^+(t - x/c) + p^-(t + x/c)\ . \tag{43}$$

Eq. (43) shows that the physical quantity p (e.g. string displacement or acoustic pressure) can be expressed as the sum of two wave quantities traveling in opposite directions. In waveguide models waves are sampled in space and time in such a way that equality holds in (42). If propagation along a one-dimensional medium, such as a cylinder, is ideal, i.e. linear, non-dissipative and non-dispersive, wave propagation is represented in the discrete-time domain by a couple of digital delay lines (Fig. 14), which propagates the wave variables p^+ and p^-.

[7] The D'Alembert solution can be derived by inserting the exponential eigenfunction e^{st+vx} into (40)

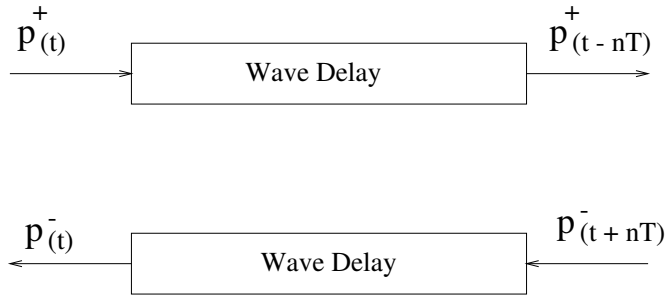

Figure 14: Wave propagation propagation in a ideal (i.e. linear, non-dissipative and non-dispersive) medium can be represented, in the discrete-time domain, by a couple of digital delay lines.

Let us consider deviations from ideal propagation due to losses and dispersion in the resonator. Usually, these linear effects are lumped and simulated with a few filters which are cascaded with the delay lines. Losses due to terminations, internal frictions, etc., give rise to gentle low pass filters, whose parameters can be identified from measurements. Wave dispersion, which is often due to medium stiffness, is simulated by means of allpass filters whose effect is to produce a frequency-dependent propagation velocity [83]. The reflecting terminations of the resonator (e.g., a guitar bridge) can also modeled as filters. In virtue of linearity and time invariance, all the filters can be condensed in a single higher-order filtering block, and all the delays can be connected to form a single longer delay line. As a result, we would get the recursive comb filter, described in chapter 3, which forms the structure of the Karplus-Strong synthesis algorithm [47].

One-dimensional waveguide models can be connected together by means of waveguide junctions, thus forming digital waveguide networks, which are used for simulation of multi-dimensional media (e.g., membranes [34]) or complex acoustic systems (e.g., several strings attached to a bridge [17]). The general treatment of waveguide networks is beyond the scope of this book [85].

Appendix A

Mathematical Fundamentals

A.1 Classes of Numbers

A.1.1 Fields

Given a set $\mathcal{F}$ of numbers, two operations called sum and product over these numbers, and some algebraic properties that we are going to enumerate, $\mathcal{F}$ is called a field. The sum of two elements of the field $u, v \in \mathcal{F}$ is still an element of the field and has the following properties:

S1, Associative Property : $(u + v) + w = u + (v + w)$

S2, Commutative Property : $u + v = v + u$

S3, Existence of the Zero : There exists one and only element in $\mathcal{F}$, called the zero, that is the neutral element for the sum, i.e., $u + 0 = u$, for all $u \in \mathcal{F}$

S4, Existence of the Opposite : For each $u \in \mathcal{F}$ there exists one and only element in $\mathcal{F}$, called the opposite of u, and written as $-u$, such that $u + (-u) = 0$.

The product of two elements of the field $u, v \in \mathcal{F}$ is still an element of the field and has the following properties:

P1, Associative Property : $(uv)w = u(vw)$

P2, Commutative Property : $uv = vu$

P3, Existence of the Unity : There exists one and only element in $\mathcal{F}$, called the unity, that is the neutral element for the product, i.e., $u1 = u$, for all $u \in \mathcal{F}$

P4, Existence of the Inverse : For each $u \in \mathcal{F}$ different from zero, there exists one and only element in $\mathcal{F}$, called the inverse of u, and written as u^{-1}, such that $uu^{-1} = 1$.

The two operations of sum and product are jointly characterized by the distributive properties:

D1, Distributive Property : $u(v + w) = uv + uw$

D2, Distributive Property : $(v + w)u = vu + wu$

The existence of the opposite and the reciprocal implies the existence of two other operations, namely, the difference $u - v = u + (-v)$ and the quotient $u/v = u(v^{-1})$.

Given the properties of a field, we can say that the natural numbers $\mathcal{N} = 0, 1, \ldots$ do not form a field since, for instance, they do not have an opposite. Similarly, the integer numbers $\mathcal{Z} = \ldots, -2, -1, 0, 1, \ldots$ do not form a field because, in general, they do not have an inverse. On the other hand, the rational numbers $\mathcal{Q}$, which are given by ratios of integers, do satisfy all the properties of a field.

The real numbers $\mathcal{R}$ are all those numbers that can be expressed in decimal notation as $x.y$, where the number of digits of y is not necessarily bounded. Real numbers can be obtained as the union of the set of rational numbers with the set of transcendental numbers, i.e., those numbers that can not be expressed as a ratio of integers. An example of transcendental number is π, which is the ratio between the circumference and the diameter of any circle. The real numbers do form a field, and the rationals are a subfield of the reals.

A.1.2 Rings

A set of numbers provided with sum and product, and such that the properties S1–4, P1 e D1–2 are satisfied is called a ring. If P2 is satisfied we have a commutative ring, and if P3 is satisfied the ring has a unity. For instance, the set $\mathcal{Z}$ of integer numbers forms a commutative ring with a unity.

Whenever we want to indicate the sets of ordered couples or triples of elements belonging to a field (or a ring) $\mathcal{F}$ we will use the notation $\mathcal{F}^2$ or $\mathcal{F}^3$, respectively.

A.1.3 Complex Numbers

The classes of numbers introduced so far are instrumental to a hierarchical system, where the natural numbers are contained in the integers, which are part of the rationals, and this latter class in contained in the real numbers. This hierarchy is resemblant of the temporal evolution of the classes of numbers since the antiquity to the XVI century. The extension of the hierarchy was always motivated by the ease with which practical and formal problems could be solved by manipulation of numerical symbols. The same kind of motivation led to the introduction of the class of complex numbers. As we will see in sec. A.3), they come into play when one wants to represent the solutions of a second-order equation.

In order to define the complex numbers, we have to define the imaginary unity i as that number that multiplied by itself (i.e., squared), gives -1. Therefore,

$$i^2 \triangleq ii = -1 \; . \tag{1}$$

In several branches of engineering the symbol j is preferred to i, because it is more easily distinguished from the symbol of current. In this book, the symbol i is used exclusively.

Given the preliminary definition of i, the complex numbers are defined as the couples

$$x + iy \tag{2}$$

where x and y are real numbers called, respectively, real and imaginary part of the complex number.

Given two complex numbers $c_1 = x_1 + iy_1$ and $c_2 = x_2 + iy_2$ the four operations are defined as follows[1]:

Sum : $c_1 + c_2 = (x_1 + x_2) + i(y_1 + y_2)$

Difference : $c_1 - c_2 = (x_1 - x_2) + i(y_1 - y_2)$

Product : $c_1 c_2 = (x_1 x_2 - y_1 y_2) + i(x_1 y_2 + x_2 y_1)$

Quotient : $\dfrac{c_1}{c_2} = \dfrac{(x_1 x_2 + y_1 y_2) + i(y_1 x_2 - x_1 y_2)}{{x_2}^2 + {y_2}^2}$.

[1]The expressions can be derived by application of the usual algebraic operations on real numbers and by substituting i^2 with -1. In order to derive the quotient, it is useful to multiply and divide by $x_2 - iy_2$.

If the introduction of complex numbers dates back to the XVI century, their geometric interpretation, that gave an intuitive framework for widespread use, was introduced in the XVIII century. The geometric interpretation is simply obtained by considering the geometric number $c = x+iy$ as a point of the plane having coordinates x and y. This interpretation, depicted in fig. 1, allows to switch from the orthogonal coordinates x and y to the polar coordinates ρ and θ, called magnitude (or absolute value) and phase (or argument), respectively. The x and y axes are called, respectively, the real and imaginary axes. The magnitude of a complex number is calculated by application of the Theorem of Pythagoras:

$$\rho^2 = x^2 + y^2 = (x + iy)(x - iy) = c\bar{c} \tag{3}$$

where $\bar{c}$ is the complex conjugate of c, also depicted in fig 1[2]. The argument of a complex number is the angle formed by the positive horizontal semi-axis with the line conducted from the geometric point to the origin of the complex plane. The argument is signed, and the sign is positive for anti-clockwise angles (see fig. 1).

A.2 Variables and Functions

In mathematics, the entities that one works with are often arbitrary elements of a class of numbers. In these cases, the entities can be represented by a variable x defined in a domain D. In this appendix, we have already used some variables implicitly, for instance, to state the properties of a field.

When the domain is an interval of the field of real numbers having extremes a and b, we can say that x is a continuous variable of the interval $[a, b]$ and we write $a \leq x \leq b$.

When every value of the variable x is associated with one and only one value of another variable y we say that y is a function of x, and we write

$$y = f(x) \ . \tag{4}$$

x is said to be the independent variable (argument) while y is the dependent variable, and the set of values that it takes for different assumed by x in its domain is called the codomain. If, for each $x_1 \neq x_2$, $f(x_1) \neq f(x_2)$, then domain and codomain have a biunivocal correspondence. In that case the roles

[2] It is easy to show that the magnitude of the product is equal to the product of the magnitudes. Vice versa, the magnitude of the sum is not equal to the sum of the magnitudes

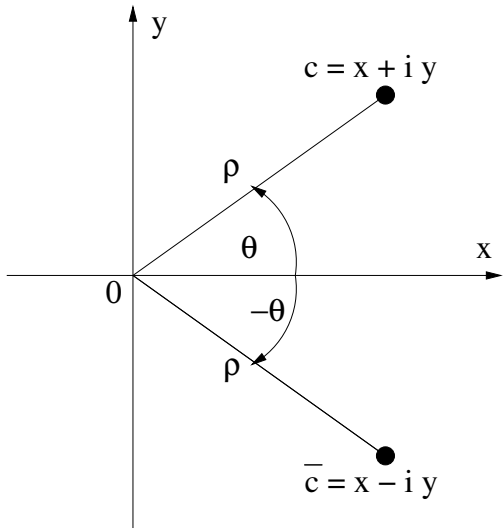

Figure 1: Geometric interpretation of a complex number

of domain and codomain can be inverted, and it is possible to define an inverse function $x = f^{-1}(y)$. In general, functions can have more than one independent variable, thus indicating a relation among many variables.

Often functions are defined by means of algebraic expressions, and associated with domains and interpretations for the variables. For instance, the pitch h (in Hz) of the note produced by an ideal string can be expressed by the function

$$h = \frac{1}{2l}\sqrt{\frac{t}{d}}\,, \tag{5}$$

where l is the length of the string in meters, t is the string tension in Newton, and d is the density per unit length (Kg/m). This concise expression allows to represent the pitch of a note whatever are the values of length, tension, and density, as long as these values belong to the domain of non-negative real numbers (indicated by $\mathcal{R}^+$).

Functions can be graphically represented in the cartesian plane. The abscissa corresponds with an independent variable, and the ordinate corresponds to the dependent variable. If we have more than one dependent variable, only one is represented in abscissa, and the other ones are set to constant values.

For example, fig. 2 shows the function (5), with values of tension and density [3] set to 952N and 0.0367Kg/m, respectively. The domain of string lengths ranges from 0.5m to 4.0m.

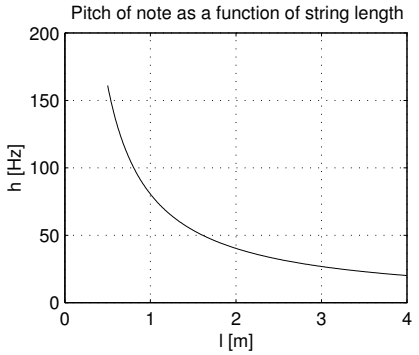

Figure 2: Pitch of a note as a function of string length

The chart of fig. 2 can be obtained by a simple script in Octave or Matlab:

```
r=0.0367; t=952;  % definitions of density and tension
l=[0.5:0.01:4.0]; % domain for the string length
h=1./(2*l)*sqrt(t/r); % expression for pitch
plot(l,h);
grid;
title('Pitch of note as a function of string length');
xlabel('l [m]');
ylabel('h [Hz]');
% replot; % Octave only
```

In order to visualize functions of two variables, we can also use three-dimensional representations. For example, the function (5) can be visualized as in fig. 3 if the variables length and tension are defined over intervals and the density is set to a constant. In such a representation, the function of two dependent variables becomes a surface in 3D. The Octave/Matlab script for fig. 3 is the following:

```
r=0.0367;         % definition of density
l=[0.5:0.1:4.0]; % domain for the string length
```

[3] These values are appropriate for the piano note C2.

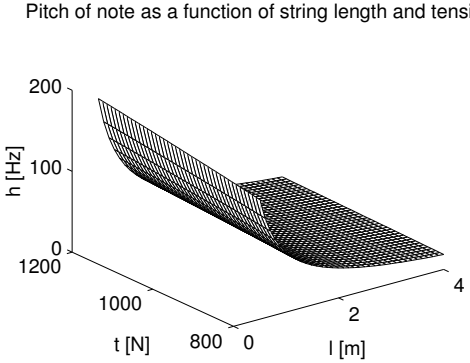

Figure 3: Pitch of a note as a function of string length and tension

```
t=[800:10:1200]; % domain for the string tension
h=(1./(2*l')*sqrt(t./r))'; % expression for pitch
mesh(l,t,h);
grid; title('Pitch of note as a function of string \
            length and tension');
xlabel('l [m]');
ylabel('t [N]');
zlabel('h [Hz]');
% replot; % Octave only
```

Of a multivariable function we can also give the contour plot, i.e., the plot of curves obtained for constant values of the dependent variable. For example, in the function (5), if we let the dependent variable to take only seven prescribed values, the cartesian plane of length and tension displays seven curves (see fig. 4). Each curve corresponds to an horizontal cut of the surface of fig. 3.

The Octave/Matlab script producing fig. 4 is the following:

```
r=0.0367;        % definition of density
l=[0.5:0.1:4.0]; % domain for the string length
t=[800:10:1200]; % domain for the string tension
h=(1./(2*l')*sqrt(t./r))'; % expression for pitch
% contour(h', 7, l, t); % Octave only
co=contour(l, t, h, 7); % Matlab only
clabel(co);              % Matlab only
title('Pitch of note as a function of string \
```

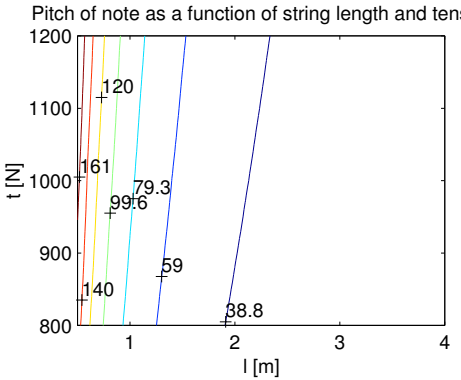

Figure 4: Contour plot of pitch as a function of string length and tension

```
       length and tension');
xlabel('l [m]');
ylabel('t [N]');
zlabel('h [Hz]');
```

A.3 Polynomials

An important class of one-variable functions is the class of polynomials, which are weighted sums of non-negative powers of the independent variable. Each power with its coefficient is called a monomial. A polynomial has the form

$$y = f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \,, \tag{6}$$

where the numbers a_i are called coefficients and, for the moment, they can be considered as real numbers. The highest power that appears in (6) is called the order of the polynomial.

The second-order polynomials, when represented in the $x - y$ plane, produce a class of curves called parabolas, while third-order polynomials generate cubic curves.

We call solutions, or zeros, or roots of a polynomial those values of the independent variable that produce a zero value of the dependent variable. For second and third-order polynomials there are formulas to derive the zeros in

closed form. Particularly important is the formula for second-order polynomials:

$$ax^2 + bx + c = 0 \tag{7}$$

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} . \tag{8}$$

As it can be easily seen by application of (8) to the polynomial $x^2 + 1$, the roots of a real-coefficient polynomial are real numbers. This observation was indeed the initial motivation for introducing the complex numbers as an extension of the field of real numbers.

The Fundamental Theorem of Algebra states that every n-th order real-coefficient polynomial has exactly n zeros in the field of complex numbers, even though these zeros are not necessarily all distinct from each other. Moreover, the roots that do not belong to the real axis of the complex plane, are couples of conjugate complex numbers.

For polynomial of order higher than three, it is convenient to use numerical methods in order to find their roots. These methods are usually based on some iterative search of the solution by increasingly precise approximations, and are often found in numerical software packages such as Octave.

In Octave/Matlab a polynomial is represented by the list of its coefficients from a_n to a_0. For instance, $1 + 2x^2 + 5x^5$ is represented by

```
p = [5 0 0 2 0 1]
```

and its roots are computed by the function

```
rt = roots(p) .
```

In this example the roots found by the program are

```
rt =
  -0.87199 + 0.00000i
   0.54302 + 0.57635i
   0.54302 - 0.57635i
  -0.10702 + 0.59525i
  -0.10702 - 0.59525i
```

and only the first one is real. If the previous result is saved in a variable `rt`, the complex numbers stored in it can be visualized in the complex plane by the directive

```
axis([-1,1,-1,1]);
plot(real(rt),imag(rt),'o');
```

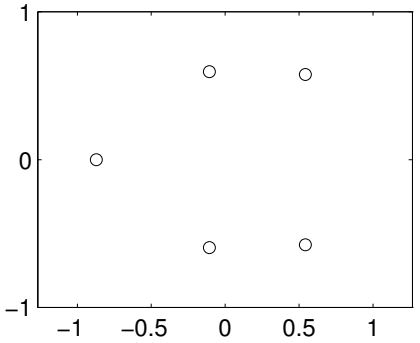

Figure 5: Roots of the polynomial $1 + 2x^2 + 5x^5$ in the complex plane

and the result is reported in fig. 5.

It can be shown that the real-coefficient polynomials form a commutative ring with unity if the operations of sum and product are properly defined. The sum of two polynomials is a polynomial whose order is the highest of the orders of the operands, and having coefficients which are the sums of the respective coefficients of the operands. The product is done by application of the usual distributive and associative properties to the product of sums of powers. The order of the product is given by the sum of the orders of the polynomial operands, and the k-th coefficient of the product is obtained by the coefficients a_i and b_j of the operands by the formula

$$c_k = \sum_{i+j=k} a_i b_j \ , \tag{9}$$

where this notation indicates a sum whose addenda are characterized by a couple of indices i, j that sum up to k.

As it can be seen from sec. 1.4, the polynomial multiplication is formally identical to the convolution of discrete signals, and this latter operation is fundamental in digital signal processing.

A.4 Vectors and Matrices

Physicists use arrows to indicate physical quantities having both an intensity and a direction (e.g., forces or velocities). These arrows, sometimes called

vectors, are oriented according to the direction of the physical quantity and their length is proportional to the intensity. These vectors can be located in the plane (or the 3D space) as if they were departing from the origin. In this way, they can be represented by the couple (or triple) of coordinates of their second extremity. This representation allows to perform the sum of vectors and the multiplication of a vector by a constant as the usual algebraic operations done with each separate coordinate:

$$\begin{aligned}(x_1, y_1, z_1) + (x_2, y_2, z_2) &= (x_1 + x_2, y_1 + y_2, z_1 + z_2) \\ \alpha(x_1, y_1, z_1) &= (\alpha x_1, \alpha y_1, \alpha z_1)\end{aligned} \tag{10}$$

More generally, an n-coordinate vector is defined in a field $\mathcal{F}$ as the ordered set of n numbers[4] $x_i \in \mathcal{F}$:

$$\mathbf{v} = [x_1, \ldots, x_n] \; . \tag{11}$$

The set of all n-coordinate vectors defined in the field $\mathcal{F}$, for which the operations (10) give vectors within the set itself, form the n-dimensional vector space $\mathcal{V}_n(\mathcal{F})$.

Every subset of $\mathcal{V}_n(\mathcal{F})$ that is closed[5] with respect to the operations (10) is called vector subspace of $\mathcal{V}_n(\mathcal{F})$. For instance, in the two-dimensional plane, the points of a cartesian axis form a subspace of the plane. Similar, subspaces of the plane are given by any straight line passing through the origin, and subspaces of the 3D space are given by any plane passing through the origin.

m vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$, are said to be linearly independent if there is no choice of m coefficients $a_1, \ldots, a_m$ (the choice of all zeros is excluded) such that

$$a_1\mathbf{v}_1 + \cdots + a_m\mathbf{v}_m = 0 \; . \tag{12}$$

In the 2D plane, two points on different cartesian axes are linearly independent, as are any two points belonging to different straight lines passing through the origin. Viceversa, points belonging to the same straight line passing through the origin are always linearly dependent.

It can be shown that, in an n-dimensional space $\mathcal{V}_n(\mathcal{F})$, every set of $m \geq n$ vectors is linearly dependent. A set of n linearly independent vectors (if they

[4] In this book, the square brackets are used to indicate vectors and matrices. This is also the notation used in Octave. Moreover, the variables representing vectors or matrices are always typed in bold font.

[5] A set I is closed with respect to an operation on its elements if the result of the operation is always an element of I.

exist) is called a basis of $\mathcal{V}_n(\mathcal{F})$, in the sense that any other vector of$\mathcal{V}_n(\mathcal{F})$ can be obtained as a linear combination of the base vectors. For instance, the vectors $[1,0,0]$, $[0,1,0]$, and $[0,0,1]$ form a basis for the 3D space, but there are infinitely many other bases.

Between any two vectors of the same vector space the operation of dot product is defined, and it returns the scalar sum of the component-by-component products. As a formula, the dot product is written as

$$\mathbf{v}'\mathbf{w} \triangleq \sum_{j=1}^{n} v_j w_j \ . \tag{13}$$

By convention, with $\mathbf{v}$ we indicate a column vector, while $\mathbf{v}'$ denotes its transposition into a row. Therefore, the operation (13) can be referred as a row-column product.

A matrix can be considered as a list of vectors, organized in a table where each element of the list occupies (by convention) one column. A matrix having n rows and m columns defined over the field $\mathcal{F}$ can be written as

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \dots & a_{1,m} \\ & \dots & \\ a_{n,1} & \dots & a_{n,m} \end{bmatrix} \in \mathcal{F}^{n \times m} \ . \tag{14}$$

The multiplication of a matrix $\mathbf{A} \in \mathcal{F}^{n \times m}$ by a (column) vector $\mathbf{v} \in \mathcal{V}_m(\mathcal{F})$ is defined as

$$\mathbf{A}\mathbf{v} = \begin{bmatrix} \sum_{j=1}^{m} a_{1,j} v_j \\ \dots \\ \sum_{j=1}^{m} a_{n,j} v_j \end{bmatrix} \ , \tag{15}$$

i.e., as a (column) vector whose i-th element is given by the dot product of the i-th row by the vector $\mathbf{v}$.

The product of a matrix $\mathbf{A} \in \mathcal{R}^{l \times m}$ by a matrix $\mathbf{B} \in \mathcal{R}^{m \times n}$ can be obtained as a list of vectors, each being the product of matrix $\mathbf{A}$ by a column of $\mathbf{B}$, and it is a matrix $\mathbf{C} \in \mathcal{R}^{l \times n}$. The product is properly defined only if the number of column of the first matrix is equal to the number of rows of the second matrix. In general, the order of factors can not be reversed, i.e., the matrix product is not commutative.

Given a matrix $\mathbf{A}$, the matrix $\mathbf{A}'$ obtained by exchanging each row with the corresponding column is called the transposed of $\mathbf{A}$.

Languages such as Octave and Matlab were initially conceived as languages for matrix manipulation. Therefore, they offer data structures and builtin operators for representing and manipulating matrices. For example, a matrix $\mathbf{A} \in \mathcal{R}^{2\times 3}$ can be represented as

```
A = [1, 2, 3; 4, 5, 6];
```

where the semicolon is used to separate one row from the following one. A column vector can be entered as

```
b = [1; 2; 3];
```

or, alternatively, we can transpose a row vector

```
b = [1, 2, 3]';
```

Given the definitions of the variables `A` and `b`, we can multiply the Matrix by the vector and assign the result to a new vector variable `c`:

```
c = A * b
```

thus obtaining the result

```
c =

  14
  32
```

The product of a matrix $\mathbf{A} \in \mathcal{R}^{l\times m}$ by a matrix $\mathbf{B} \in \mathcal{R}^{m\times n}$ is represented by

```
A * B
```

When we want to do element-wise operations between two or more vectors or matrices having the same size, we just have to place a dot before the operator symbol. For instance,

```
[1, 2, 3] .* [4, 5, 6]
```

returns the (row) vector `[4 10 18]` as a result.

Octave allows to operate on scalars, vectors, and matrices belonging to the complex field, just by representing as a sum of real and imaginary parts (e.g., `2 + 3i`).

When we use Octave/Matlab to handle functions, or to draw their plot, we usually operate on collections of points that are representative of the functions. There is a concise way to assign to a variable all the values regularly spaced (with step `inc`) between a `min` and a `max`:

```
x = [min, inc, max];
```

This kind of instruction has been used to plot the function of fig. 2. After having defined the domain as the vector of points

```
l=[0.5: 0.1: 4.0];
```

the vector representing the codomain has been computed by application of the

function to the vector `l`:

```
f=1./(2*l)*sqrt(t/r);
```

A.4.1 Square Matrices

The n-th order square matrices defined over a field $\mathcal{F}$ are a set $\mathcal{F}^{n\times n}$ which is very important for its affinity with the classes of numbers. In fact, for these matrices the sum and product are always defined and it is easy to verify that the properties S1–4, P1, and D1–2 of appendix A.1 do hold. The property P3 is also verified and the neutral element for the product is found in the unit diagonal matrix, which is a matrix that has ones in the main diagonal[6] and zeros elsewhere. In general, the commutativity is not ensured for the product, and a matrix might not admit an inverse matrix, i.e., an inverse obeying to property P4. In the terminology introduced in appendix A.1, the square matrices $\mathcal{F}^{n\times n}$ form a ring with a unity. This observation allows us to treat the square matrices with compact notation, as a class of numbers which is not much different from that of integers[7].

A.5 Exponentials and Logarithms

Given a number $a \in \mathcal{R}^+$, it is clear what is its natural m-th power, that is the number obtained multiplying a by itself m times. The rational power $a^{1/m}$, with m a natural number, is defined as the number whose m-th power gives a. If we extend the power operator to negative exponents by reciprocation of the positive power, we give meaning to all powers a^r, with r being any rational number. The extension to any real exponent is obtained by imposing continuity to the power function. Intuitively, the function $f(x) = a^x$ describes a continuous curve that "interpolates" the values taken at the points where x is rational. The power operator has the following fundamental properties:

E1 : $a^x a^y = a^{x+y}$

E2 : $\dfrac{a^x}{a^y} = a^{x-y}$

[6] The main diagonal goes from the top leftmost corner to the bottom rightmost corner.

[7] Two important differences with the ring of integers is the non commutativity and the possibility that two non-zero matrices multiplied together give the zero matrix (the zero matrix admits non-zero divisors).

E3 : $(a^x)^y = a^{xy}$

E4 : $(ab)^x = a^x b^x$.

The function $f(x) = a^x$ is called exponential with base a.

Given these preliminary definitions and properties, we define the logarithm of y with base a

$$x = \log_a y \ , \tag{16}$$

as the inverse function of $y = a^x$. In other words, it is the exponent that must be given to the base in order to get the argument y. Since the power a^x has been defined only for $a > 0$ and it gives always a positive number, the logarithm is defined only for positive values of the independent variable y.

Logarithms are very useful because they translate products and divisions into sums and differences, and power operations into multiplications. Simply stated, by means of the logarithms it is possible to reduce the complexity of certain operations. In fact, the properties E1–3 allow to write down the following properties:

L1 : $\log_a xy = \log_a x + \log_a y$

L2 : $\log_a \dfrac{x}{y} = \log_a x - \log_a y$

L3 : $\log_a x^y = y \log_a x$.

In sound processing, the most interesting logarithm bases are 10 and 2. The base 10 is used to define the decibel (symbol dB) as a ratio of two quantities. If the quantities x and y are proportional to sound pressures (e.g., rms level), we say that x is wdB larger than y if $x > y > 0$ and

$$w = 20 \log_{10} \frac{x}{y} \ . \tag{17}$$

When the quantities x and y are proportional to a physical power (or intensity), their ratio in decibel is measured by using a factor 10 instead of 20[8] in (17).

The base 2 is used in all branches of computer sciences, since most computing systems are based upon binary representations of numbers (see the appendix A.9). For instance, the number of bits that is needed to form an address in a memory of 1024 locations is

$$\log_2 1024 = 10 \ . \tag{18}$$

[8] In acoustics [86], the power is proportional to the square of a pressure. Therefore, applying property L3, we fall back into the definition (17).

In Octave/Matlab, the logarithms of `x` having base 2 and 10 are indicated with `log2(x)` and `log10(x)`, respectively. Fig. 6 shows the curves of the logarithms in base 2 and 10. From these curves we can intuitively infer how, in any base, $\log 1 = 0$, and how the function approaches $-\infty$ (minus infinity) as the argument approaches zero.

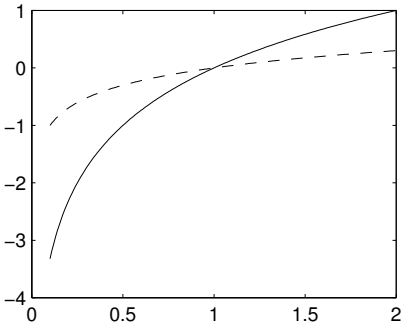

Figure 6: Logarithms expressed in the fundamental bases 2 (solid line) and 10 (dashed line)

Given a logarithm expressed in base a, it is easy to convert it in the logarithm expressed in another base b. The formula that can be used is

$$\log_b x = \frac{\log_a x}{\log_a b} \,. \tag{19}$$

A base of capital importance in calculus is the Neper number e, a transcendental number approximately equal to 2.7183. As we will see in appendix A.7.1, the exponentials expressed in base e are eigenfunctions for the derivative operator. In other words, differential linear operators do not alter the form of these exponentials. Moreover, the exponential with base e admits an elegant translation into an infinite series of addenda

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots \,, \tag{20}$$

where $n!$ is the factorial of n and is equal to the product of all integers ranging from 1 to n. It can be proved that the infinite sum on the right-hand side of (20) gives meaning to the exponential function even where its argument is complex.

A.6 Trigonometric Functions

Trigonometry describes the relations between angles and segments subtended by these angles. The main trigonometric functions are easily visualized on the complex plane, as in fig. 7, where the unit circle is explicitly represented.

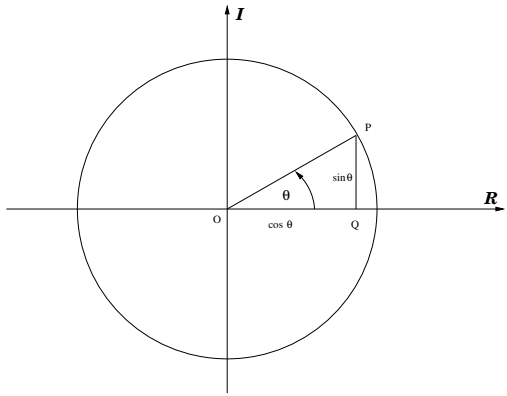

Figure 7: Trigonometric functions on the complex plane

An angle θ cuts on the unit circle an arc whose length is defined as the measure in radians of the angle. Since the circumference has length 2π, the 360^o angle measures 2π radians, and the 90^o angle corresponds to $\pi/2$ radians. The main trigonometric functions are:

Sine $\sin\theta = \overline{PQ}$

Cosine $\cos\theta = \overline{OQ}$

Tangent $\tan\theta = \overline{PQ}/\overline{OQ}$

It is clear from fig. 7 and from the Pythagoras' theorem that, for any θ, the identity

$$\sin^2\theta + \cos^2\theta = 1 \tag{21}$$

is valid.

The angle, considered positive if oriented anti clockwise, can be considered the independent variable of trigonometric functions. Therefore, we can use Octave/Matlab to plot the main trigonometric functions, thus obtaining fig. 8. These plots can be obtained as subplots of a same figure by the following Octave/Matlab script:

```
theta = [0:0.01:4*pi];
s = sin(theta);
c = cos(theta);
t = tan(theta);
subplot(2,2,1); plot(theta,s);
axis([0,4*pi,-1,1]);
grid; title('Sine of an angle');
xlabel('angle [rad]');
ylabel('sin');
% replot; % Octave only
subplot(2,2,2); plot(theta,c);
grid; title('Cosine of an angle');
xlabel('angle [rad]');
ylabel('cos');
% replot; % Octave only
subplot(2,2,3); plot(theta,t);
grid; title('Tangent of an angle');
xlabel('angle [rad]');
ylabel('tan');
axis([0,4*pi,-6,6]);
% replot; % Octave only
```

It is clear from the plots that the functions sine and cosine are periodic with period 2π, while the function tangent is periodic with period π. Moreover, the codomain of sine and cosine is limited to the interval $[-1, 1]$, while the codomain of the tangent takes values on all real axis. The tangent approaches infinity for all the values of the argument that multiples of $\pi/2$, i.e. in these points we have vertical asymptotes.

As we can see from fig. 7, a complex number c, having magnitude ρ and argument θ, can be represented in its real and imaginary parts as

$$c = x + iy = \rho \cos\theta + i\rho \sin\theta \; . \tag{22}$$

A fundamental identity, that links trigonometry with exponential functions, is the Euler formula

$$e^{i\theta} = \cos\theta + i \sin\theta \; , \tag{23}$$

which expresses a complex number laying on the unit circumference as an exponential with imaginary exponent[9]. When θ is left free to take any real value, the exponential (23) generates the so-called complex sinusoid.

[9]The actual meaning of the exponential comes from the series expansion (20)

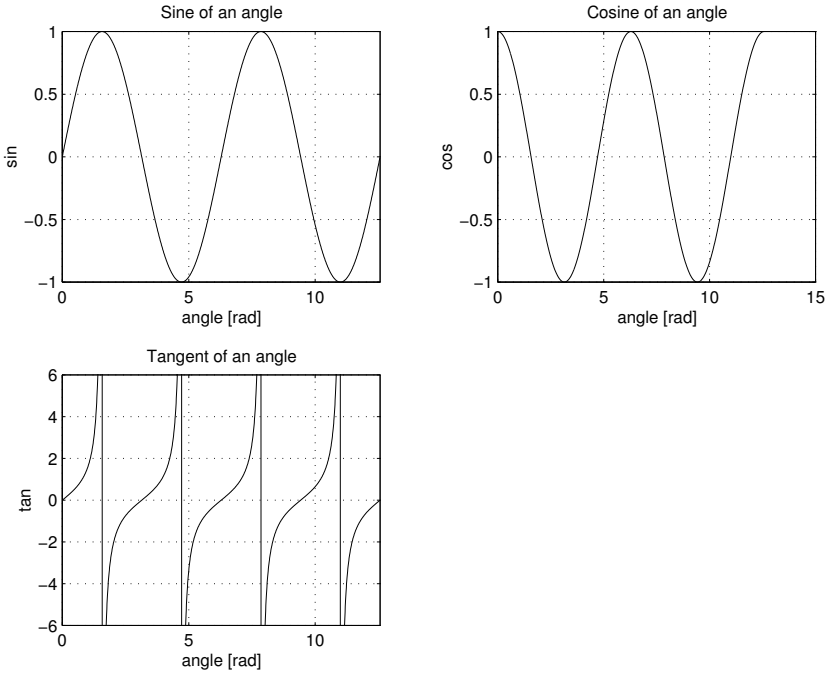

Figure 8: Trigonometric functions

Any complex number c having magnitude ρ and argument θ can be represented in compact form as

$$c = \rho e^{i\theta} \ , \tag{24}$$

and to it we can apply the usual rules of power functions. For instance, we can compute the m-th power of c as

$$c^m = \rho^m e^{im\theta} = \rho^m (\cos m\theta + i \sin m\theta) \ , \tag{25}$$

thus showing that it is obtained by taking the m-th power of the magnitude and multiplying by m the argument. The (25) is called De Moivre formula.

The order-m root of a number c is that number b such that $b^m = c$. In general, a complex number admits m order-m distinct complex roots[10]. The De

[10]For instance, 1 admits two square roots (1 and -1) and four order-4 roots (1, -1, i, -i).

Moivre formula establishes that[11] the order-m roots of 1 are evenly distributed along the unit circumference, starting from 1 itself, and they are separated by a constant angle $2\pi/m$.

At this point, we propose some problems for the reader:

- Prove the following identities, which are corollaries of the Euler identity

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2} , \tag{26}$$

$$\sin\theta = \frac{e^{i\theta} - e^{-i\theta}}{2i} . \tag{27}$$

- Prove the "most beautiful formula in mathematics" [59]

$$e^{i\pi} + 1 = 0 . \tag{28}$$

- Prove, by means of the De Moivre formula, the following identities:

$$\cos 2\theta = \cos^2\theta - \sin^2\theta , \tag{29}$$

$$\sin 2\theta = 2\sin\theta\cos\theta . \tag{30}$$

- Prove, by the representation of unit-magnitude complex numbers $e^{i\theta}$, that the following identities are true:

$$\cos(\theta + \phi) = \cos\theta\cos\phi - \sin\theta\sin\phi , \tag{31}$$

$$\sin(\theta + \phi) = \cos\theta\sin\phi + \sin\theta\cos\phi . \tag{32}$$

A.7 Derivatives and Integrals

A.7.1 Derivatives of Functions

Given the function $y = f(x)$ (for the moment, we only consider functions of one variable), it might be interesting to find the places where local maxima and minima are located. It is natural, in such a search, to focus on the slope of

[11] The reader is invited to justify this statement by an example. The simplest non-trivial example is obtained by considering the cubic roots of 1.

the line that is tangent to the function curve, in such a way that local maxima and minima are found where the slope of the tangent is zero (i.e., the tangent is horizontal). This operation is possible for all regular functions, which are functions without discontinuities and without sharp corners. Given this assumption of regularity, the shape of the curve can be defined at any point, thus becoming itself a function of the same independent variable. This function is called derivative and is indicated with

$$y' = \frac{dy}{dx} \,. \tag{33}$$

The notation (33) recalls how the local shape of a curve can be computed: the tangent line is drawn, two distinct points are taken on this line, the ratio between the *d*ifferences of coordinates y and x of the points is formed. As we have already seen in appendix A.6, this operation corresponds to the computation of the trigonometric tangent, whose argument is the angle formed by the tangent line with the horizontal axis. This observation should have made the terminology more clear.

In fig. 9 the polynomial $y = f(x) = 4 + 3x + 2x^2 - x^3$ is plotted for $x \in [-4, 4]$, together with its derivative. As we can see, the derivative is positive where $f(x)$ is increasing, negative where $f(x)$ is decreasing, and zero where $f(x)$ has a local extremal point.

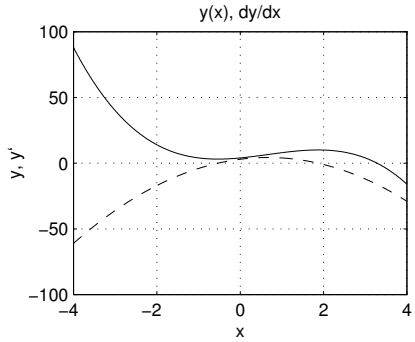

Figure 9: A degree-3 polyonomial and its derivative

The Octave/Matlab script used to produce fig. 9 is the following:

```
x = [-4:0.01:4];          % domain
poli = [-1 2 3 4];        % coefficients of a degree-3
```

```
                          % polynomial
y = polyval(poli, x);     % evaluation of the polynomial
% coefficients of the derivative of the polynomial
% polid = polyderiv(poli); % Octave only
polid = poli(1:length(poli)-1).*[length(poli)-1:-1:1];
                          % Matlab only
                          % (polyderiv is not available)
yp = polyval(polid, x); % evaluation of the derivative
plot(x, y, '-'); hold on;
plot(x, yp, '--'); hold off;
ylabel('y, y'');
xlabel('x');
title('y(x), dy/dx');
grid;
% replot; % Octave only
```

In the script there are two new directives. The first one is the function invocation `polyval(poli, x)`, which returns the vector of values taken by the polynomial, whose coefficients are specified in `poli`, in correspondence with the points specified in `x`. The second directive is the function invocation `polideriv(poli)`, which returns the coefficient of the polynomial that is the derivative of `poli`. This function is not available in Matlab, but it can be replaced by an explicity calculation, as indicated in the script. The fact that the derivative of a polynomial is still a polynomial is ensured by the derivation rules of calculus. Namely, the derivative of a monomial is a lower-degree monomial given by the rule

$$\frac{d(ax^n)}{dx} = anx^{n-1} \; . \tag{34}$$

The derivative is a linear operator, i.e.,

- The derivative of a sum of functions is the sum of the derivatives of the single functions
- The derivative of a product of a function by a constant is the product of the constant by the derivative of the function

Another important property of the derivative is that it transforms the composition of functions in a product of functions. Given two functions $y = f(x)$ and $z = g(y)$, the composed function $z = g(f(x))$ is obtained by replacing

the domain of the second function with the codomain of the first one [12]. The derivative of the composed function is expressed as

$$\frac{dz}{dx} = g'(y)f'(x) = \frac{dz}{dy}\frac{dy}{dx} , \tag{35}$$

which remarks the effectiveness of the notation introduced for the derivatives.

For the purpose of this book, it is useful to know the derivatives of the main trigonometric functions, which are given by

$$\frac{d\sin x}{dx} = \cos x \tag{36}$$

$$\frac{d\cos x}{dx} = -\sin x \tag{37}$$

$$\frac{d\tan x}{dx} = \frac{1}{\cos^2 x} \tag{38}$$

Therefore, we can say that a sinusoidal function conserves its sinusoidal character (it is only translated along the x axis) when it is subject to derivation. This property comes from the fact, already anticipated, that the exponential with base e is an eigenfunction for the derivative operator, i.e.,

$$\frac{de^x}{dx} = e^x . \tag{39}$$

If we consider the complex exponential e^{ix} as the composition of an exponential function with a monomial with imaginary coefficient, it is possible to apply the linearity of derivative to the composed function and derive the formulas (36) and (37).

In order to derive (38) we also have to know the rule to derive quotients of functions. In general, products and quotients of functions are derived according to

$$\frac{d\,[f(x)g(x)]}{dx} = f'(x)g(x) + f(x)g'(x) \tag{40}$$

$$\frac{d\,[g(x)/f(x)]}{dx} = \frac{g'(x)f(x) - f'(x)g(x)}{f^2(x)} . \tag{41}$$

[12] For instance, $\log x^2$ is obtained by squaring x and then taking the logarithm or, by the property L3 of logarithms, ...

A.7.2 Integrals of Functions

For the purpose of this book, it is sufficient to informally describe the defined integral of a function $f(x), x \in \mathcal{R}$ as the area delimited by the function curve and the horizontal axis in the interval between two edges a e b (see fig. 10). When the curve stays below the axis the area has to be considered negative, and positive when it stays above the axis. The defined integral is represented in compact notation as

$$\int_a^b f(x)dx \ , \tag{42}$$

and it takes real values.

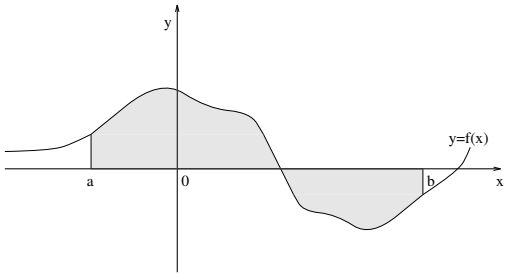

Figure 10: Integral defined as an area

In order to compute an integral we can use a limiting procedure, by approximating the curve with horizontal segments and computing an approximation of the integral as the sum of areas of rectangles. If the segment width approaches zero, the computed integral converges to the actual measure.

There is a symbolic approach to integration, which is closely related to function derivation. First of all, we observe that for the integrals the properties of linear operators do hold:

- The integral of a sum of functions is the sum of integrals of the single functions
- The integral of a product of a function by a constant is the product of the constant by the integral of the function.

Then, we generalize the integral operator in such a way that it doesn't give a single number but a whole function. In order to do that, the first integration

edge is kept fixed, and the second one is left free on the x axis. This newly defined operator is called indefinite integral and is indicated with

$$F(x) = \int_a^x f(u)du \ . \tag{43}$$

The argument of function $f()$, also called integration variable, has been called u to distinguish it from the argument of the integral function $F()$.

The genial intuition, that came to Newton and Leibniz in the XVII century and that opened the way to a great deal of modern mathematics and science, was that derivative and integral are reciprocal operations and, therefore, they are reversible. This idea is translated in a remarkably simple formula:

$$F'(x) = f(x) \ , \tag{44}$$

which is valid for regular functions. The reader can justify the (44) intuitively by thinking of the derivative of $F(x)$ as a ratio of increments. The increment at the numerator is given by the difference of two areas obtained by shifting the right edge by dx. The increment at the denominator is dx itself. Called m the average value taken by $f()$ in the interval having length dx, such value converges to $f(x)$ as dx approaches zero.

$F(x)$ is also called a primitive function of $f(x)$, where the article *a* subtends the property that indefinite integrals can differ by a constant. This is due to the fact that the derivative of a constant is zero, and it justifies the fact that the position of the first integration edge doesn't come into play in the relationship (44) between a function and its primitive.

At this point, it is easy to be convinced that the availability of a primitive $F(x)$ for a function $f(x)$ allows to compute the definite integral between any two edges a and b by the formula

$$\int_a^b f(u)du = F(b) - F(a) \ . \tag{45}$$

We encourage the reader to find the primitive functions of polynomials, sinusoids, and exponentials. To acquire better familiarity with the techniques of derivation and integration, the reader without a background in calculus is referred to chapter VIII of the book [25].

A.8 Transforms

The analysis and manipulation of functions can be very troublesome operations. Mathematicians have tried to find alternative ways of expressing func-

tions and operations on them. This research has expressed some transforms which, in many cases, allow to study and manipulate some classes of functions more easily.

A.8.1 The Laplace Transform

The Laplace Transform was introduced in order to simplify differential calculus. The Laplace transform of a function $y(t)$, $t \in \mathcal{R}$ is defined as a function of the complex variable s:

$$Y_L(s) = \int_{-\infty}^{+\infty} y(t)e^{-st}dt, s \in \Gamma \subset \mathcal{C} \ , \tag{46}$$

where Γ is the region where the integral is not divergent. The region Γ is always a vertical strip in the complex plane, and within this strip the transform can be inverted with

$$y(t) = \frac{1}{2\pi j} \int_{\sigma-j\infty}^{\sigma+j\infty} Y_L(s)e^{st}ds, t \in \mathcal{R} \ . \tag{47}$$

The edges of the integration (47) indicate that the integration is performed along a vertical line with abscissa σ.

Example 1. The most important transform for the scope of this book is that of the causal complex exponential function, which is defined as

$$y(t) = \begin{cases} e^{s_0 t} & t \geq 0 \ , s_0 \in \mathcal{C} \\ 0 & t < 0 \end{cases} \ . \tag{48}$$

Such transform is calculated as[13]

$$\begin{aligned} Y_L(s) &= \int_{-\infty}^{+\infty} y(t)e^{-st}dt = \int_0^{+\infty} e^{s_0 t}e^{-st}dt = \int_0^{+\infty} e^{-(s-s_0)t}dt = \\ &= -\frac{1}{s-s_0}(e^{-(s-s_0)\infty} - e^{-(s-s_0)0}) = \frac{1}{s-s_0} \ , \end{aligned} \tag{49}$$

and it is convergent for those values of s having real part that is larger than the real part of s_0. We have seen in appendix A.7 that the exponential function is an eigenfunction for the operators derivative and integral, which are fundamental for the description of physical systems. Therefore, we can easily understand the practical importance of the transform (49).

[13] In a rigorous treatment, the notation $e^{-(s-s_0)\infty}$ should be replaced by a limiting operation for $t \to \infty$.

###

A central property of the Laplace transform is given by the transformation of the derivative operator into a multiply by s:

$$\frac{dy(t)}{dt} \leftrightarrow sY_L(s) - [y(0)] \ , \tag{50}$$

where the term within square brackets is the initial value in the case that $y(t)$ is a causal function, i.e. $y(t) = 0$ for any $t < 0$. Conversely, the integral is converted into a division by the complex variable s:

$$\int_{-\infty}^{t} y(u)du \leftrightarrow \frac{1}{s}Y_L(s) \ . \tag{51}$$

Since physics describes systems by means of equations containing derivatives and integrals, these equations can be transformed into polynomial equations by means of the Laplace transform, and the calculus turns out to be simplified.

Example 2. The second Newton's law states that, for a body having mass m, the relationship among force f, mass, acceleration a, displacement x, and time t, can be expressed by

$$f = ma = m\frac{d^2x}{dt^2} \ , \tag{52}$$

where the notation $\frac{d^2x}{dt^2}$ indicates a second derivative, i.e. the derivative applied twice. The relation (52) is Laplace-transformed into the polynomial equation

$$F_L(s) = s^2 mX_L(s) - [smx(0) + mx'(0)] \ , \tag{53}$$

where the term within square brackets is determined by the initial condition of displacement and velocity at time 0.

###

A.8.2 The Fourier Transform

The Fourier transform of $y(t)$, $t \in \mathcal{R}$, can be obtained as a specialization of the Laplace transform in the case that the latter is defined in a region comprising the imaginary axis. In such case we define[14]

$$Y(\Omega) \triangleq Y_L(j\Omega) \ , \tag{54}$$

[14]Often the Fourier transform is defined as a function of f, where $2\pi f = \Omega$

or, in detail,

$$Y(\Omega) = \int_{-\infty}^{+\infty} y(t) e^{-j\Omega t} dt \ , \tag{55}$$

where $j\Omega$ indicates a generic point on the imaginary axis. Since the kernel of the Fourier transform is the complex sinusoid (i.e., the complex eponential) having radial frequency Ω, we can interpret each point of the transformed function as a component of the frequency spectrum of the function $y(t)$. In fact, given a value $\Omega = \Omega_0$ and considered a signal that is the complex sinusoid $y(t) = e^{j\Omega_1 t}$, the integral (55) is maximized when choosing $\Omega_0 = \Omega_1$, i.e., when $y(t)$ is the complex conjugate of the kernel [15]. The codomain of the transformed function $Y(\Omega)$ belongs to the complex field. Therefore, the spectrum can be decomposed in a magnitude spectrum and in a phase spectrum.

A.8.3 The Z Transform

The domains of functions can be classes of numbers of whatever kind and nature. If we stick with functions defined over rings, particularly important are the functions whose domain is the ring of integer numbers. These are called discrete-variable functions, to distinguish them from functions of variables defined over $\mathcal{R}$ or $\mathcal{C}$, which are called continuous-variable functions.

For discrete-variable functions the operators derivative and integral are replaced by the simplest operators difference and sum. This replacement brings a new definition of transform for a function $y(n)$, $n \in \mathcal{Z}$:

$$Y_Z(z) = \sum_{n=-\infty}^{+\infty} y(n) z^{-n}, z \in \Gamma \subset \mathcal{C} \ . \tag{56}$$

The transform (56) is called Z transform and the region of convergence is a ring[16] of the complex plane. Within this ring the transform can be inverted.

Example 3. The Z transform of the discrete-variable causal exponential

[15] Exercise: find the Fourier transform of the causal complex exponential (48), with $s_0 = \alpha + j\Omega_0$, and show that it has maximum magnitude for $\Omega = \Omega_0$.

[16] A ring here is the area between two circles and not an algebraic structure.

is[17]

$$\begin{aligned} Y_Z(z) &= \sum_{n=-\infty}^{+\infty} y(n) z^{-n} = \sum_{n=0}^{+\infty} e^{z_0 n} z^{-n} = \\ &= \sum_{0}^{+\infty} (e^{z_0} z^{-1})^n = \frac{1}{1 - e^{z_0} z^{-1}} , \end{aligned} \tag{57}$$

and it is convergent for values of z that are larger than $e^{\Re(z_0)}$ in magnitude[18].

Similarly to what we saw for continuous-variable functions, the Fourier transform for discrete-variable functions can be obtained as a specialization of the Z transform where the values of the complex variable are restricted to the unit circumference.

$$Y(\omega) = Y_Z(e^{j\omega}) , \tag{58}$$

or, in detail,

$$Y(\omega) = \sum_{n=-\infty}^{+\infty} y(n) e^{-j\omega n} . \tag{59}$$

In this book, we use the symbol ω for the radian frequency in the case of discrete-variable functions, leaving Ω for the continuous-variable functions.

###

A.9 Computer Arithmetics

A.9.1 Integer Numbers

In order to fully understand the behavior of several hardware and software tools for sound processing, it is important to know something about the internal representation of numbers within computer systems. Numbers are represented as strings of binary digits (0 and 1), but the specific meaning of the string depends on the conventions used. The first convention is that of unsigned integer

[17] The latter equality in (57) is due to the identity $\sum_{n=0}^{+\infty} a^n = \frac{1}{1-a}$, $|a| < 1$, which can be verified by the reader with $a = 1/2$.

[18] $\Re(x)$ is the real part of the complex number x

numbers, whose value is computed, in the case of 16 bits, by the following formula

$$x = \sum_{i=0}^{15} x_i \times 2^i \ , \tag{60}$$

where x_i is the i-th binary digit starting from the right. The binary digits are called bits, the rightmost digit is called least significant bit (LSB), and the leftmost digit is called the most significant bit (MSB). For instance, we have

$$0100001100100110_2 = 2^1 + 2^2 + 2^5 + 2^8 + 2^9 + 2^{14} = 17190 \ , \tag{61}$$

where the subscript 2 indicates the binary representation, being the usual decimal representation indicated with no subscript.

The leftmost bit is often interpreted as a sign bit: if it is set to one it means that the sign is minus and the absolute value is given by the bits that follow. However, this is not the representation that is used for the signed integers. For these numbers the two's complement representation is used, where the leftmost bit is still a sign bit, but the absolute value of a negative number is recovered by bitwise complementation of the following bits, interpretation of the result as a positive integer, and addition of one. For instance, with four bits we have

$$1010_2 = -(0101_2 + 1) = -(5 + 1) = -6 \ . \tag{62}$$

The two's complement representation has the following advantages:

- there is only one representation of the zero[19].
- it has a cyclic structure: a unit increment of the largest representable positive number gives the negative number with the largest absolute value
- the sums between signed numbers are performed by simple bitwise operation and without caring about the sign (a carry on the left can be ignored)

We note that

- the negative number with the largest absolute value is $100\ldots0_2$. Its absolute value exceeds that of the largest positive number (i.e., $011\ldots1_2$) by one
- the negative number with the smallest absolute value is represented by $111\ldots1_2$

[19] Vice versa, the sign and magnitude representation has one positive and one negative zero

- the range of the numbers representable in two's complement with 16 bits is $[-2^{15}, 2^{15} - 1] = [-32768, 32767]$
- the range of the numbers representable in two's complement with 8 bits is $[-2^7, 2^7 - 1] = [-128, 127]$

Often, in computer memory words and addresses are organized as collections of 8-bit packets, called bytes. Therefore, it is useful to use a representation where the bits are considered in packets of four units, each packet tacking integer values from 0 to 15. This representation is called hexadecimal and, for the numbers between 10 and 15, it uses the hexadecimal "digits" A, B, C, D, E, F. For instance, a 16-bit binary number can be represented as

$$0100101100100110_2 = 4B26_{16} \ . \tag{63}$$

A.9.2 Rational Numbers

We have two alternative possibilities to represent rational non-integer numbers:

- fixed point
- floating point

The fixed point representation is similar to the representation of integer numbers, with te difference that we have a decimal point at a prescribed position. The digits are divided into two sets: the integer part and the fractional part. The 16-bit representation, without sign and with 3 bits of integer part is

$$x = \sum_{i=-13}^{2} x_i \times 2^i \ , \tag{64}$$

and is obtained by multiplication of the integer number on 16 bits by 2^{-13}. In the two's complement representation, the operations can be done without caring of the position of the decimal point, as we would be operating on integer numbers. Often, the rational numbers are considered to be normalized to one, i.e., to be limited to the range $[-1, 1)$. In such a case, the decimal point is placed before the leftmost binary digit.

For the floating point representation we can follow different conventions. In particular, the IEEE 754 floating-point single-precision numbers obey to the following rules

- the number is represented as

$$1.xx\ldots x_2 \times 2^{yy\ldots y_2} , \tag{65}$$

  where x are the binary digits of the mantissa and y are the binary digits of the exponent

- The number is represented on 32 bits according to the following block decomposition
  - bit 31: sign bit
  - bits 23–30: exponent $yy\ldots y$ in biased representation[20], from the most negative $00\ldots0$ to the most positive $11\ldots1$
  - bits 0–22: mantissa in unsigned binary representation

The IEEE 754 standard of double-precision floating-point numbers uses 11 bits for the exponent and 52 bits for the mantissa.

It should be clear that both the fixed- and the floating-point representations take a subset of rational numbers. Fixed-point numbers are equally spaced between the minimum and the maximum representable value with a quantization step equal to 2^{-d}, where d is the number of digits on the right of the decimal point. Floating-point numbers are unevenly distributed, being more sparse for large values of the exponent and more dense for little exponents. Floating-point numbers have the possibility to represent a large range, from 2×10^{-38} to 2×10^{38} in single precision, and from 2×10^{-308} to 2×10^{308} in double precision. Therefore, it is possible to do many computations without worrying of errors due to overflow. Moreover, the high density of small numbers reduces the problems due to the quantization step. This is paid in terms of a more complicated arithmetics.

[20]The bias is 127. Therefore, the exponent 1 is coded as $1 + 127 = 128 = 10000000_2$. The biased representation simplifies the bit-oriented sorting operations.

Appendix B

Tools for Sound Processing

(with Nicola Bernardini)

Audio signal processing is essentially an engineering discipline. Since engineering is about practical realizations the discipline is best taught using real-world tools rather than special didactic software. At the roots of audio signal processing there are mathematics and computational science: therefore we strongly recommend using one of the advanced maths softwares available off the shelf. In particular, we experienced teaching with Matlab, or with its Free Software counterpart Octave [1]. Even though much of the code can be ported from Matlab to Octave with minor changes, there can still be some significant advantage in using the commercial product. However, Matlab is expensive and every specialized toolbox is sold separately, even though an less-expensive student edition is available. On the other hand, Octave is free software distributed under the GNU public license. It is robust, highly integrated with other tools such as Emacs for editing and GNUPlot for plotting.

For actual sound applications, there are at least three other categories of softwares for sound synthesis that it is worth considering: languages for sound processing, interactive graphical building environments, and inline sound editors.

When sound applications are targeted to the market of information appliances, it is likely that the processing algorithms will be implemented on low-cost hardware specifically tailored for typical signal-processing operations.

[1] http://www.octave.org

Therefore, it is also useful to look at how signal-processing chips are usually structured.

B.1 Sounds in Matlab and Octave

In Octave/Matlab, monophonic sounds are simply one-dimensional vectors (rows or columns), so that they can be transformed by means of matrix algebra, since vectors are first–class variables. In these systems, the computations are vectorized, and the gain in efficiency is high whenever looped operations on matrices are transformed into compact matrix-algebra notation [9]. This peculiarity is sometimes difficult to assimilate by students, but the theory of matrices needed in order to start working is really limited to the basic concepts and can be condensed in a two-hours lecture.

Processing in Octave/Matlab usually proceeds using monophonic sounds, as stereo sounds are simply seen as couples of vectors. It is necessary to make clear what the sound sample rate is at each step, i.e., how many samples are needed to produce one second of sound.

Let us give an example of how we can create a $440Hz$ sinusoidal sound, lasting 2 seconds, and using the sample rate $F_s = 44100Hz$:

```
f = 440; % pitch in Hz
Fs= 44100; % sample rate in Hz
l = 2;   % soundlength in seconds
Y = sin(2*pi*f/Fs*[0:Fs*l]); % sound vector
```

The sound is simply defined by application of the function `sin()` to a vector of `Fs*l + 1` elements (namely, 88200 elements) containing an increasing ramp, suitably scaled so that f cycles are represented in Fs samples.

Once the sound vector has been defined, one may like to listen to it. On this point, Matlab and Octave present different behaviors, also dependent on the machine and operating system where they are running. Matlab offers the function `sound()` that receives as input the vector containing the sound and, optionally, a second parameter indicating the sample rate. Without the second parameter, the default sample rate is 8192Hz. Up to version 4.2 of Matlab, the number of reproduction bits was 8 on a Intel-compatible machine. More recent versions of Matlab reproduce sound vectors using 16 bits of sample resolution. In order to reproduce the sound that we have produced with the above script we should write

```
sound(Y, Fc);
```

Up to now, in the core Octave distribution the function that allows to produce sounds from the Octave interpreter is `playaudio()`, that can receive "filename" and "extension" as the first and second argument, respectively. The extension contains information about the audio file format, but so far only the formats raw data linear and mu-law are supported. Alternatively, the argument of `playaudio` can be a vector name, such as `Y` in our example. The reproduction is done at 8 bits and 8192 Hz, but it would be easy to modify the function so that it can use better quantizations and sample rates. Fortunately, there is the octave-forge project [2] that contains useful functions for Octave which are not in the main distribution. In the audio section we notice the following interesting functions (quoting from the help lines):

`sound(x [, fs])` Play the signal through the speakers. Data is a matrix with one column per channel. Rate fs defaults to 8000 Hz. The signal is clipped to [-1, 1].

`soundsc(x, fs, limit)` or `soundsc(x, fs, [ lo, hi ])` Scale the signal so that [min(x), max(x)] → [-1, 1], then play it through the speakers at 8000 Hz sampling rate. The signal has one column per channel.

`[x, fs, sampleformat] = auload('filename.ext')` Reads an audio waveform from a file. Returns the audio samples in data, one column per channel, one row per time slice. Also returns the sample rate and stored format (one of ulaw, alaw, char, short, long, float, double). The sample value will be normalized to the range [-1,1) regardless of the stored format. This does not do any level correction or DC offset correction on the samples.

`ausave('filename.ext', x, fs, format)` Writes an audio file with the appropriate header. The extension on the filename determines the layout of the header. Currently supports .wav and .au layouts. Data is a matrix of audio samples, one row time step, one column per channel. Fs defaults to 8000 Hz. Format is one of ulaw, alaw, char, short, long, float, double

B.1.1 Digression

In Matlab versions older than 5, the function `sound` had a bug that is worth analyzing because it sheds some light on risks that may be connected with the

[2] http://www.sourceforge.net

internal representations of integer numbers. Let us construct a sound as a casual sequence of numbers having values 1 and -1:

```
Fs = 8192;
W=rand(size(0:Fs)) - 0.5;
for i = 1:length(W)
  if (W(i)>0) W(i) = 1.0;
  else W(i) = -1.0;
  end;
end;
```

In order to be convinced that such sound is a spectrally-rich noise we can plot its spectrum, that would look like that of fig. 1.

Surprisingly enough, in old Matlab versions on Intel-compatible architectures if the sound `W` was played using `sound(W)` the audio outcome was, at most, a couple of clicks corresponding to the start and end transients.

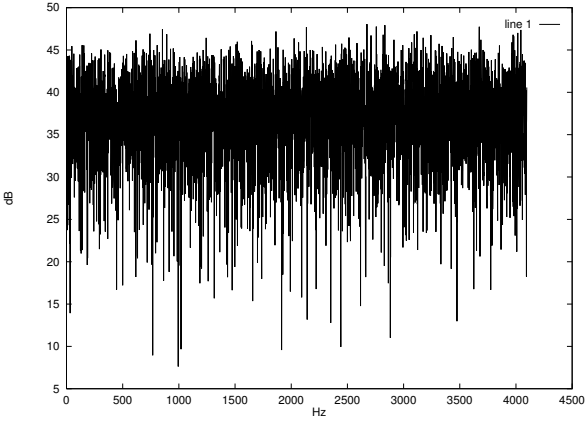

Figure 1: Spectrum of a random 1 and -1 sequence

This can be explained by thinking that, on 8 bits, 256 quantization levels can be represented. A number between -1.0 and $+1.0$ is recasted into the 8-bits range by taking the integer part of its product by 128. The problem is that, when the resulting integer number is represented in two's complement, the number $+1.0$ is not representable since, on 8 bits, the largest positive number that can be represented is 127. Due to the circularity of two's complement representation, the multiplication 1.0×128 produces the number -128, which

is also the representation of -1.0. Therefore, the audio device sees a constant sequence of numbers equal to the most negative representable number, and it does not produce any sound, except for the transients due to the initial and final steps. Once the problem had been discovered and understood, the user could circumvent it by rescaling the signal in a slightly larger range, e.g., `[-1, 1.1]`.

In the Matlab environment the acquisition and writing of sound files from and to the disk is done by means of the functions `auread()`, `auwrite()`, `wavread()`, e `wavwrite()`. The former couple of functions work with files in `au` format, while the latter couple work with files in the popular `wav` format. In earlier version of Malab (before version 5) these functions only dealt with 8-bit files, thus precluding high-quality audio processing. For users of old Matlab versions, two routines are available for reading and writing 16-bit `wav` files, called wavr16.m and wavw16.m, written by F. Caron and modified to ensure Octave compatibility. An example of usage for `wavr16()` is

```
[L,R,format] = wavr16('audiofile.wav')
```

that returns the right and left channels of the file `audiofile.wav`, in the `L` and `R` vectors, respectively. The two vectors are identical if the file is monophonic. The returned vector `format` has four components containing format information: the kind of encoding (indeed only PCM linear is recognized), the number of channels, the sample rate, and the number of quantization bits.

An example of invocation of the function `wavw16()` is

```
wavw16('audiofile.wav', M, format)
```

where `format` is, again, a four-component vector containing format information, and `M` is a one- or two-column matrix containing the channels to be written in a monophonic or stereophonic file.

Since sounds are handled as monodimensional vectors, sound processing can be reduced in most cases to vectorial operations. The iterative, sample-by-sample processing is quite inefficient with interpreters such as Octave or Matlab, that are optimized to handle matrices. As an example of elementary processing, consider a simple smoothing operation, obtained by substitution of each input sound sample with the average between itself and the following sample. Here is a script that does this operation in Octave, after having loaded a monophonic sound file:

```
[L,R,format] = wavr16('ma1.wav');
S = (L + [L(2:length(L)); 0]) / 2; %``smoothed'' sound
```

The operation is expressed in a very compact way by summation of the vector `L` with the vector itself left-shifted by one position[3]. The smoothing operation may be expressed iteratively as follows:

```
[L,R,format] = wavr16('ma1.wav');
S = L/2;
for i=1:length(L)-1
 S(i) = (L(i) + L(i+1))/2;
end;
```

The code turns out to be less compact but, probably, more easily understandable. However, the running time is significantly higher because of the `for` loop.

In the Matlab environment, there is a collection of functions called the Signal Processing Toolbox. In the examples of this book we do not use those functions, preferring public-domain routines written for Octave, possibly modified to be usable within Matlab. One such function is function is stft.m, that allows to have a time-frequency representation of a signal. This can be useful for time-frequency processing and representation, as in the script

```
SS = stft(S);
mesh(20*log10(SS));
```

whose result is a 3D representation of the time-frequency behavior of the sound contained in `S`.

B.2 Languages for Sound Processing

In this section we briefly show how sounds are acquired and processed using languages that have been explicitely designed for sound and music processing.

The most widely used language is probably `Csound`, developed by Barry Vercoe at the Massachusetts Institute of Technology and available since the middle eighties. `Csound` is a direct descendant of the family of Music-N languages that was created by Max Mathews at the Bell Laboratories since the late fifties. In this family, the language of choice for most computer-music composers between the sixties and the eighties was `Music V`, that established a standard in symbology of basic operators, called Unit Generators (UG).

[3]The last element is set to zero to fill the blank left by the left-shift operation on `L`. The reader can extend the example in such a way that the input sound is overlapped and summed with its echo delayed by 200ms.

According to the Music-N tradition, the UGs are connected as if they were modules of an analog synthesizer, and the resulting patch is called an instrument. The actual connecting wires are variables whose names are passed as arguments to the UGs. An orchestra is a collection of instruments. For every instrument, there are control parameters which can be used to determine the behavior of the instrument. These parameters are accessible to the interpreter of a score, which is a collection of time-stamped invocations of instrument events (called notes). Fig. 2 shows a schematic description of how Music-V-like languages work: `a)` is a Music-V source text[4] while `b)` is its graphical representation. The orchestra/score metaphor, the decomposition of an orchestra

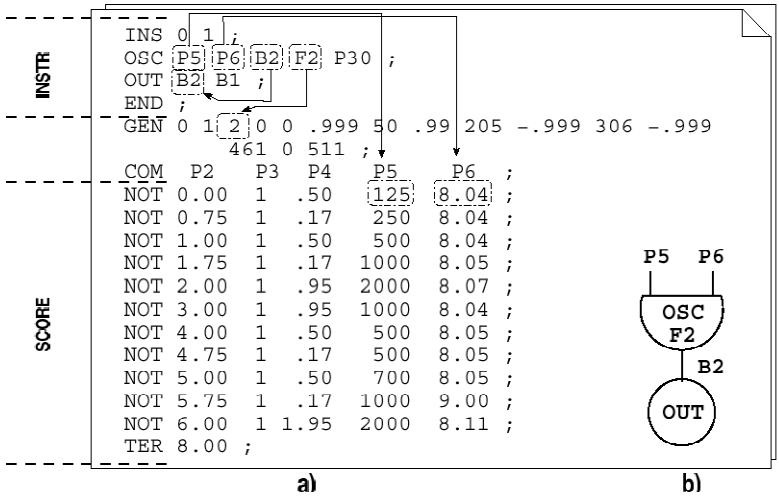

Figure 2: Music-V file description

into non-interacting instruments, and the description of a score as a sequence of notes, are all design decisions which were taken in respect of a traditional view of music. However, many musical and synthesis processes do not fit well in such a metaphorical frame. As an example, consider how difficult it is to express modulation processing effects that involve several notes played by a single synthesis instrument (such as those played within a single violin bow-

[4] picked up from [56, page 45]

ing): it would be desirable to have the possibility of modifying the instrument state as a result of a chain of weakly synchronized events (that is, to perform some sort of per-thread processing). Instead, languages such as Music V rely on special initialization steps encoded within instruments to handle articulatory gestures involving several pitches.

Other models have been proposed for dealing with less rigid descriptions of sound and music events. One such model is tied to the Nyquist language [5], developed by the team of Roger Dannenberg at the Carnegie Mellon University [28]. This language provides a unified treatment of music and sound events and is based on functional programming (Lisp language). Algorithmic manipulations of symbols, processing of signals, and structured temporal modifications are all possible without leaving a consistent framework. In particular, Nyquist exploits the idea of behavioral abstraction, i.e. time-domain transformations are interpreted in an abstract sense and the details are encapsulated in descriptions of behaviors [27]. In other words, musical concepts such as duration, onset time, loudness, time stretching, are specified differently in different UGs. Modern compositional paradigms benefit from this unification of control signals, audio signals, behavioral abstractions and continuous transformations.

Placing some of the most widely used languages for sound manipulation along an axis representing flexibility and expressiveness, the lower end is probably occupied by Csound while the upper one is probably occupied by Nyquist. Another notable language which lies somewhere in between is Common Lisp Music [6] (CLM), which was developed by Bill Schottstaedt as an extension of Common Lisp [87]. If CLM is not too far from Nyquist (thanks to the underlying Lisp language) there is another language closer to the other edge of the axis, which represents a "modernization" of Csound. The language is called SAOL [7] and it has been adopted as the formal specification of Structured Audio for the MPEG-4 standard [107]. SAOL orchestras and scores can be translated into C language by means of the software translator SFRONT [8] developed by John Lazzaro and John Wawrzynek at UC Berkeley.

The simple examples that we are presenting in this book are written in Csound, and realizations in CLM and SAOL are presented for comparison.

[5] http://www.cs.cmu.edu/~rbd/nyquist.html
[6] http://www-ccrma.stanford.edu/software/clm/
[7] http://www.saol.net
[8] http://www.cs.berkeley.edu/~lazzaro/sa/

B.2.1 Unit generator

The UGs are primitive modules that produce, modify, or acquire audio or control signals. For audio signal production, particularly important primitives are those that read tables (`table`) and run an oscillator (`oscil`), while for producing control signals the envelope generators (`line`) are important. For sound modification, there are UGs for digital filters (`reson`) and time-domain processing, such as delays (`delay`). For sound acquisition, there are special UGs (`soundin`).

According to the Music-V tradition, several UGs can be connected to form complex instruments. The connections are realized by means of variables. In Csound the instruments are collected in a file called orchestra. The instrument parameters can be initialized by arguments passed at invocation time, called p-fields. Invocations of events on single instruments are considered to be notes, and they are collected in a second file, called score. The dichotomy between orchestra and score, as well as the subdivision of the orchestra into autonomous non-interacting entities called instruments, are design choices derived from a rather traditional view of music composition. We have already mentioned how certain kinds of operation with synthesis instruments do not fit well in this view.

The way the control and communication variables are handled in instruments made of several UGs is another crucial aspect to understand the effectiveness of a computer-music language. In Csound, variables are classified as: audio-rate variables and control-rate variables. The former can vary at audio rate, the latter are usually band-limited to a lower rate. In this way it is possible to update the control variables at a lower rate, thus saving some computations. Following the treatment of Roads [78], such run-time organization is called block-oriented computation, as opposed to sample-oriented computation. This is not to say that block-oriented computation are vectorized, or intrinsically parallel on data blocks, but rather that control variables are not loaded in the machine registers at each audio cycle.

The split of variables between audio rate and control rate does not offer any semantic benefit for the composer, but it is only a way to reach higher computation speeds. Vice versa, sometimes the sound designer is forced to choose a control rate equal to the audio rate in order to avoid some artifacts. Namely, this occurs in computational structures with delayed feedback loops[9].

[9]Consider the case, pointed out to my attention by Gianantonio Patella, of a CSound instrument with a feedback delay line. Since the UG `delay` is iterated seamlessly for a number of times equal to the ratio between sample rate and control rate, the effective length of the delay turns out to be

On the other hand, vectorized computations are an alternative way to arrange the operations, that in many cases can lead to compact and efficient code, as it was shown in the smoothing example of section B.1.

In the languages that we are considering there are UGs for time-frequency processing that operate on a frame basis. Typically, the operations on a single frame can be vectorized and we can have block-oriented computations when the control rate coincides with the frame rate.

Csound also presents a third family of variables, the initialization variables, whose value is computed only when a note starts in the score. In order to partially overcome the problems of articulation between different notes, Csound allows to hold a note (`ihold`), in such a way that the following note of the same instrument can be treated differently during the initialization (`tigoto`). For instance, these commands can be used to implement a smooth transition between notes, as in a *legato*.

An interesting aspect that has to be considered is how the sound-processing languages acquire pre-recorded material for processing. In Csound there is the primitive `soundin` that acquires the samples from an audio file for as long as the note that invoked the instrument remains active. Alternatively, with the function table statement `f` a table can be loaded with the content of an audio file, and such table can be read later on by UGs such as `table` or `oscil`. This strategy allows to perform important modifications, such as transposition, stretching, grain extraction, already at the reading stage.

The Csound architecture, largely inherited from Music V, is more oriented toward sound synthesis than sound manipulation. For instance, a reverb continues to produce meaningful signal even when its input has ceased to be active, and this fact has produced the practice to call the UG `reverb` by means of a separate instrument that takes its input from global orchestra variables. On the other hand, in CLM sound transformations are more clearly stated since any filter can have a sound file name among its parameters. For CLM, a reverb is any filter whose invocation is made explicit as an argument of a sound-generation function.

B.2.2 Examples in Csound, SAOL, and CLM

Let us face the problem of reading an audio fragment memorized in the file "march.aiff" and to process it by means of a linearly-increasing transposition a 100ms echo.

A Csound solution is found in the following orchestra and score files:

extended by such number of samples.

```
; sweep.orc
        sr = 22000        ;audio rate
        kr = 220          ;control rate
        ksmps = 100       ;audio rate / control rate
        nchnls = 1        ;number of channels

        instr 1           ;sound production
ilt     = ftlen(1)/sr     ;table length in samples
kfreq   line     1, p3, p4
        ;linear envelope from 1 to p4 in p3 seconds
gas     loscil   25000, kfreq/ilt, 1, 1/ilt, 0, 1, \
                 ftlen(1)
        ;frequency-varying oscillator on table 1
        endin

        instr 2           ;sound processing
as      delay    gas, p4 ;p4 seconds delay on global
                         ;variable gas
        out p5*as + gas  ;input + delayed and
                         ;attenuated signal
        endin

; sweep.sco
; table stored from sound file
; # time size       file       skip  format chan
f 1  0   1048576 1 "march.aiff" 0      0      1
; p1 p2 p3  p4  p5
i 1  0  25  2.0      ;sound-production note
i 2  0  25  0.1 1.0  ;sound-processing note
```

The code can be easily understood by means of the comments and by reference to the Csound manual [106]. We only observe that both the sound production and processing are activated by means of notes on different instruments. The communication between the two instruments is done by means of the global variable `gas`. The audio file is preliminarly loaded in memory by means of the statement `f` of the score file. The table containing the sound file is then read by instrument 1 using the UG `loscil`, that is a sort of sampling device where the reading speed and iteration points (loops) can be imposed.

To understand how SAOL is structurally similar to CSound but syntactic-

ally more modern, we propose some SAOL code for the same solution to our processing problem. The orchestra is

```
global {
  outchannels 1;
  srate 22000;
  krate 220;

  table tabl(soundfile, -1, "march.aiff");

  route (bus1, generator);
                //   delay     amplitude
  send (effect;        0.1,        1.0; bus1);
  }

instr generator(env_ext) {
// env_ext: target point of the linear envelope
// (from 1 to env_ext)

  ksig freq;
  asig signa;
  imports table tabl;
  ivar lentab;

  lentab = ftlen(tabl)/s_rate; //table length in seconds

  freq = kline(1, dur, env_ext);
  signa = oscil(tabl, freq/lentab, 1);
  output(signa);
}

instr effect(del,ampl) {
// del: echo delay in seconds
// ampl: amplitude of the echo
  asig signa;

  signa = delay(input, del);
  output(input + ampl*signa);
}
```

while the score reduces to the line

```
 0.00
generator 25.0 2.0
```

In SAOL, variable names, parameters, and instruments are handled more clearly. The block enclosed by the keyword `global` contains some features shared by all instruments in the orchestra, such as the sample and control rate, or the audio files that are accessed by means of tables. Moreover, this section contains a configuration of the audio busses where signal travels. In the example the `generator` instrument sends its output to the bus called `bus1`. From here, signals are sent to the `effect` unit together with the processing parameters `del` and `ampl`, In the `global` section it is possible to program arbitrarily-complex paths among production and processing units.

Let us examine how the same kind of processing can be done in CLM. Here we do not have an orchestra file, but we compose as many files as there are generation or processing instruments. Every instrument is defined by means of the LISP macro `definstrument`, and afterwords it can be compiled and loaded within the LISP environment as a primitive function. The code segment that is responsible for audio sample generation is enclosed within the `Run` macro, that is expanded into C code at compilation time. In the past, the `Run` macro could also generate code for the fixed-point Digital Signal Processor (DSP) Motorola 56000, that was available in NeXT computers, in order to speed up the computations. In contemporary general-purpose computers there is no longer an advantage in using DSP code, as the C-compiled functions are very efficient and they do not suffer from artifacts due to fixed-point arithmetics.

Here is the CLM instrument that reads an audio file at variable speed:

```
(definstrument sweep (file &key
        ;; parameters:
        ;; DURATION of the audio segment to be
        ;;    acquired (seconds)
        ;; AMPSCL: amplitude scaling
        ;; FREQ-ENV: frequency envelope
  (duration 1.0) (ampscl 1.0) (freq-env
                    '(0 1.0 100 1.0)) )
  (let ((f (open-input file)))
              ;; input file assigned to variable f
     (let*
       ((beg 0)    ;; initial inst.
```

```
    (end (+ beg
            (floor (* sampling-rate duration))))
                        ;; final inst.
    (freq-read-env
            (make-env :envelope freq-env ))
                        ;; freq. env.
    (sr-convert-a
            (make-resample :file f :srate 1.0 ))
    ;; sr-convert-a:
    ;;   var. containing the acquired file
       (out-sig-a 0.0) )           ;; dummy var.
  (Run
    (loop for i from beg to end do
    (setf out-sig-a
           (* ampscl (resample sr-convert-a
                        (env freq-read-env))))
        ;; transposition envelope (in octaves)
    (outa i out-sig-a)
    (if *reverb* (revout i out-sig-a))
))))
```

The reader can notice how, within the parentheses that follow the instrument name (`sweep`), there are mandatory parameters, such as the `file` to be read, and optional parameters, such as `duration`, `ampscl`, and `freq-env`. For the optional parameters a default value is given. It is interesting how several kinds of objects can be used as parameters, namely strings (`file`), numbers (`duration, ampscl`), or envelopes with an arbitrary number of segments (`freq-env`).

The intermediate code section contains various definitions of variables and objects used by the instrument. In this section envelopes and UGs are prepared to act as desired. The `Run` section contains a `loop` that is iterated for a number of times equal to the samples to be produced. This loop contains the signal processing kernel. The read at increasing pace is performed by the UG `resample`, whose reading step is governed by the envelope passed as a parameter. The last code line sends the signal to the post-processing unit `reverb`, when that is present. In our example, the post-processing unit is a second instrument, called `eco`:

```
(definstrument eco
  (startime dur &optional (volume 1.0) (length 0.1))
```

```
(let*  (
        (d1 (make-zdelay (* sampling-rate length)))
        (vol volume)
        (beg 0)
        (end (+ beg (floor (* dur sampling-rate)))))
   (run
    (loop for i from beg to end do
      (outa i  (* vol (zdelay d1 (revin i))))
      ))
   ))
```

The `eco` instrument will have to be compiled and loaded as well. After, the entire processing will be activated by

```
(with-sound (:reverb eco :reverb-data(1.0 0.1))
       (sweep "march.wav" :duration 25 :freq-env
                                   '(0 0.0 100 1.0)))
```

The macro `with-sound` operates a clear distinction between sound production and modification, as any kind of modification is considered as a reverb.

The three sound-processing examples written in CSound, CLM, and SAOL produce almost identical results[10]. The resulting sound waveshape and its sonogram are depicted in fig. 3. This figures has been obtained by means of the analysis program `snd`, a companion program of CLM (see section B.4). From the sonogram we can visually verify that the audio file is read at increasing speed and that such read does not contain discontinuities.

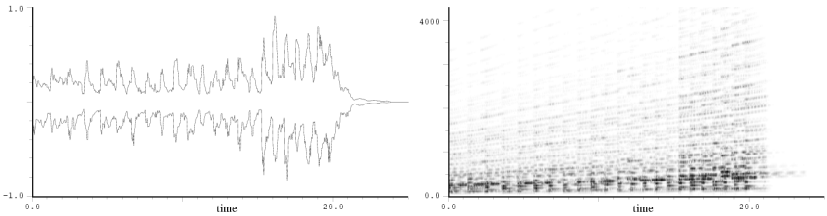

Figure 3: Waveshape and sonogram of a sound file that is echoed and read at increasing speed

[10]Subtle differences are possible due to the diversity of implementation of the UGs.

B.3 Interactive Graphical Building Environments

In recent times, several software packages have been written to ease the task of designing sound synthesis and processing algorithms. Such packages make extensive use of graphical metaphors and object abstraction reducing the processing flow to a number of small boxes with zero, one or more audio/control inputs and outputs connected by lines, thus replicating once again the old and well known modular synthesizer interface taxonomy.

The steady increase in performance of modern computers has allowed the interactive use of these graphical building environments, that become effectively rapid prototyping tools. The speed of modern processors allow sophisticated signal computations at a rate faster than the sampling rate. For instance, if the sampling rate is $F_s = 44.1$kHz, it is possible that the processor is capable to produce one or more sound samples in a time quantum $T = 1/F_s = 22.6\mu$sec. If such condition holds, even the languages of section B.2 can be used for real-time processing, i.e., they can produce an audio stream directly into the analog-to-digital converters. The user may alter this processing by control signals introduced by external means, such as MIDI messages[11].

Initially, many interactive graphical building packages where created to tame the daunting task of writing specialized code for dedicated signal processing tasks. In these packages, each object would contain some portion of DSP assembly code or microcode which would be loaded on-demand in the appropriate DSP card. With a graphical interface the user would easily construct, then, complex DSP algorithms with detailed controls coming from different sources (audio, MIDI, sensors, etc.). Several such applications still exist and are fairly widely used in the live-electronics music field (just to quote a few of the latest (remaining) ones): the Kyma/Capybara environment written by Carla Scaletti and Kurt Hebel [12], the ARES/MARS environment [7, 11, 21, 6] developed by IRIS-Bontempi, and the Scope package produced by the german firm Creamware [13].

While these specialized packages for music composers and sound designers are bound to disappear with the rapid and manifold power increase of general purpose processors[14], the concept of graphic object-oriented abstraction to eas-

[11] MIDI (Musical Instrument Digital Interface) is a standard protocol for communication of musical information

[12] http://www.symbolicsound.com

[13] http://www.creamware.de

[14] This is not a personal but rather a classic darwinian consideration: the maintenance costs of such packages added to the intrinsinc tight binding of such code with rapidly obsolescent hardware exposes them to an inevitable extinction.

ily visually construct signal processing algorithms has spur an entire new line of software products.

The most widespread one is indeed the Max package suite conceived and written by Miller Puckette at IRCAM. Born as a generic MIDI control logic builder, this package has known an enormous expansion in its commercial version produced by Cycling '74 and maintained by Dave Zicarelli [15]. A recent extension to Max, written by Zicarelli, is MSP which features real-time signal processing objects on Apple PowerMacs (i.e. on general-purpose RISC architectures). Another interesting path is being currently followed by Miller Puckette himself who is the principal author of Pure Data (`pd`) [71], an open-source public domain counterpart of Max which handles MIDI, audio and graphics (extensions by Mark Danks [16]). `pd` is developed keeping the actual processing and its graphical display as two cooperating separate processes, thus enhancing portability and easily modeling its processing priorities (sound first, graphics later) on the underlying operating system thread/task switching capabilities. `pd` is currently a very early-stage work-in-progress but it already features most of the graphic objects found in the experimental version of Max plus several audio signal processing objects. Its tcl/tk graphical interface makes its porting extremely easy (virtually "no porting at all")[17].

B.3.1 Examples in ARES/MARS and `pd`

While the use of systems that are based on specialized digital signal processors is fading out in the music and sound communities, those kinds of chips still play a crucial role in communication and embedded systems. In general, wherever one needs signal processing capabilities at a very low cost, digital signal processors come into play, with their corollary of peculiar assembly language and parallel datapaths. For this reason, it is useful to look at the ARES/MARS workstation as a prototypical example of such systems, and to see how our problem of sound echoing and continuous transposition would have been solved with such system.

In the IRIS ARES/MARS workstation there is a host computer, that is used to program the audio patches and the control environments, a micro-controller that uses its proprietary real-time operating system to handle the control signals, and one or more digital signal processors that are used to process the

[15] http://www.cycling74.com

[16] http://www.danks.org/mark/GEM/

[17] *Pure Data* currently runs on Silicon Graphics workstations, on Linux boxes and on Windows NT platforms; sources and binaries can be found at http://crca.ucsd.edu/~msp/software.html

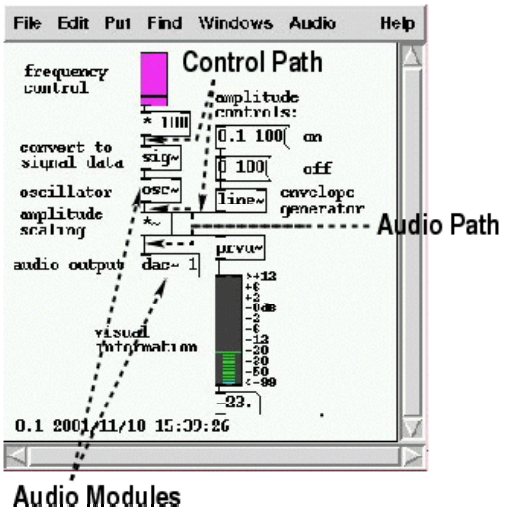

Figure 4: A `Pd` screen shot

signals at audio rate. The audio patch that solves our processing problem is shown in fig. 5. The input signal is directly taken from an analog-to-digital converter, and the output signal is sent to a digital-to-analog converter.

There are two main blocks: the first, called `HARMO`, is responsible for input signal transposition. The second, having a small clock as an icon, produces the echo. Since we want a gradually-increasing transposition, the `HARMO` block is controlled by a slowly-varying envelope, updated at a lower rate, programmed to ramp from `trasp_iniziale` to `trasp_finale`. The transposed signal goes into the delay unit and produces the echo that gets summed to the transposed signal itself before being sent to the output. Among the parameters of the `HARMO` and delay units, there are those responsible for memory management, since both units use memory buffers that must be properly allocated, as explained in section B.5.

Figure 6 shows a possible solution to our sweep-and-echo problem using `pd`. Again, we have a `harmo` block that performs the pitch transposition. However, in `pd` this harmonizer is not a native module, but it is implemented in a separate patch by means of cross-fading delay lines [110]. Similarly, the `ramped_phase` block encapsulates the operations necessary to perform a one-pass read of the wavetable containing the sound file. The subgraph in the

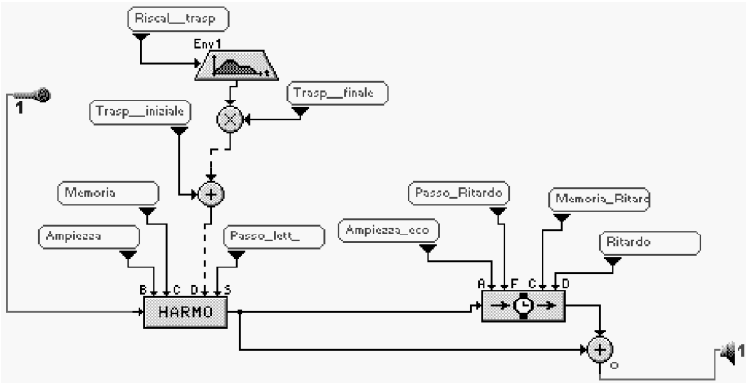

Figure 5: MARS patch for echoing and linearly-increasing transposition

lower right corner represents the linear increase in pitch transposition, obtained by means of the `line` UG and used by the `harmo` unit.

B.4 Inline sound processing

A completely different category of music software deals with inline sound processing. The software included in this category implies direct user control over sound on several levels, from its inner microscopic details up to its full external form.

In its various forms, it allows the user to: (i) process single or multiple sounds (ii) build complex sound structures into a sound stream (iii) view different graphical representations of sounds. Hence, the major difference between this category and the one outlined in the preceding paragraphs lies perhaps in this software's more general usage at the expense of less 'inherent' musical capabilities: as an example, the difference between single event and event organization (the above-mentioned *orchestra/score metaphor* and other organizational forms) which is pervasive in the languages for sound processing hardly exists in this category. However, this software allows direct manipulation of various sound parameters in many different ways and is often indispensable in musical pre-production and post-production stages.

Compared to the Music-N-type software the one of this category belongs to a sort of "second generation" computer hardware: it makes widespread and intensive use of high-definition graphical devices, high-speed sound-dedicated

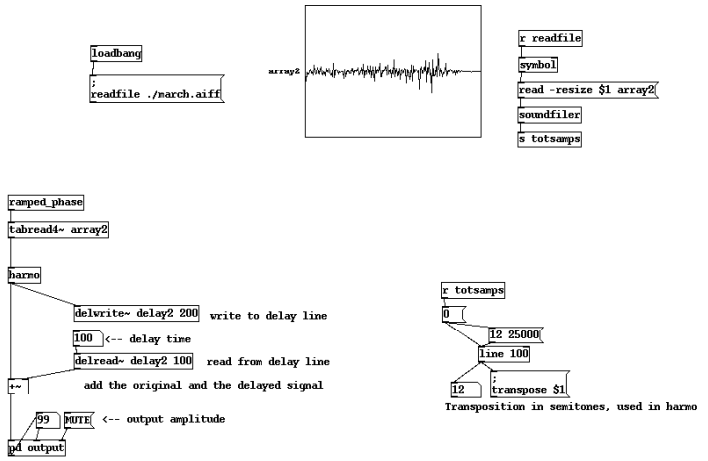

Figure 6: `pd` patch for echoing and linearly-increasing transposition

hardware, large core memory, large hard disks, etc. . In fact, we will shortly show that the most hardware-intensive software in music processing - the digital live-electronics real-time control software - belongs to one of the subcategories exposed below.

B.4.1 Time-Domain Graphical Editing and Processing

The most obvious application for inline sound processing is that of graphical editing of sounds. While text data files lend themselves very conveniently to musical data description, high-resolution graphics are fundamental to this specific field of applications where single-sample accuracy can be sacrificed to a more intuitive sound event global view.

Most graphic sound editors allow to splice and process sound files in different ways.

As fig. 7 [18] shows the typical graphical editor displays one or more soundfiles in the time-domain, allowing to modify it with a variety of tools. The important concepts in digital audio editing can be summarised as follows:

[18] The editor in this example is called *Audacity*, an Free Software audio editing and processing application written by Dominic Mazzoni, Roger Dannenberg et al.[57] (http://audacity.sourceforge.net) for Unix, Windows and MacOs workstations.

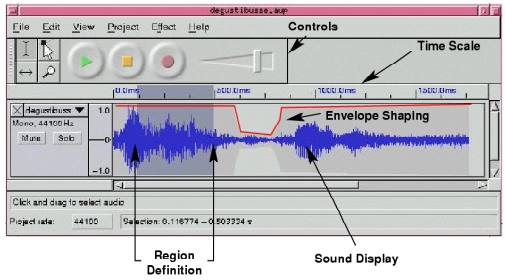

Figure 7: A typical sound editing application

- regions - these are graphically selected portions of sound in which the processing and/or splicing takes place;
- in-core editing versus window editing - while simpler editors load the sound in RAM memory for editing, the most professional ones offer buffered on-disk editing to allow editing of sounds of any length: given the current storage techniques, high-quality sound is fairly expensive in terms of storage (ca. 100 kbytes per second and growing), on-disk editing is absolutely essential to serious editing;
- editing and rearranging of large soundfiles can be extremely expensive in terms of hardware resources and hardly lend themselves to the general editing features that are expected by any multimedia application: *multiple-level undos*, *quick trial-and-error*, *non-destructive editing*, etc.: several techniques have been developed to implement these features - the most important one being the *playlist*, which allows soundfile editing and rearranging without actually touching the soundfile itself but simply storing pointers to the beginning and end of each region. As can be easily understood, this technique offers several advantages being extremely fast and non-destructive;

In fig. 8, a collection of soundfiles is aligned on the time axis according to a playlist indicating the starting time and duration of each soundfile reference (i.e. a pointer to the actual soundfile). Notice the on-the-fly amplitude rescaling of some of the soundfiles[19]

Graphical sound editors are extremely widespread on most hardware platforms: while there is no current favourite application, each platform sports one

[19] *ProTools*© is manufactured by Digidesign (http://www.digidesign.com)

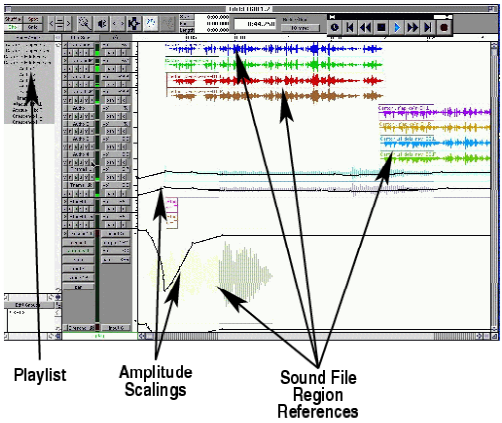

Figure 8: A snapshot of a typical *ProTools*© editing session

or more widely used editors which may range from the US$ 10000 professional editing suites for the Apple Macintosh to the many Free Software programs for unix workstations. In the latter category, it is worthwile to mention the *snd* application by Bill Schottstaedt [20] which features a back-end processing in CLM. More precisely, sounds and commands can be exchanged back and forth between CLM and snd, in such a way that the user can choose at any time the most adequate between inline and language-based processing.

B.4.2 Analysis/Resynthesis Packages

Analysis/Resynthesis packages belong to a closely related but substantially different category: they are generally medium-sized applications which offer different editing capabilities. These packages are termed *analysis/resynthesis* packages because editing and processing is preceded by an analysis phase which extracts the desired parameters in their most significant and convenient form; editing is then performed on the extracted parameters in a variety of ways and after editing, a resynthesis stage is needed to re-transform the edited parameters into a sound in the time domain. In different forms, these applications do: (i) perform various types of analyses on a sound (ii) modify the analysis data (iii) resynthesize the modified analysis.

[20] http://www-ccrma.stanford.edu/software/snd/

Many applications feature a graphical interface that allows direct editing in the frequency-domain: the prototypical application in this field is *Audiosculpt* developed by Philippe Depalle, Chris Rogers and Gilles Poirot at the IRCAM [21] (Institut de Recherche et Coordination Acoustique-Musique) for the Apple Macintosh platform. Based on a versatile FFT-based phase vocoder called *SVP*(which stands for *Super Vocodeur de Phase*), Audiosculpt is essentially a drawing program which allows the user to "draw" on the spectrum surface of a sound.

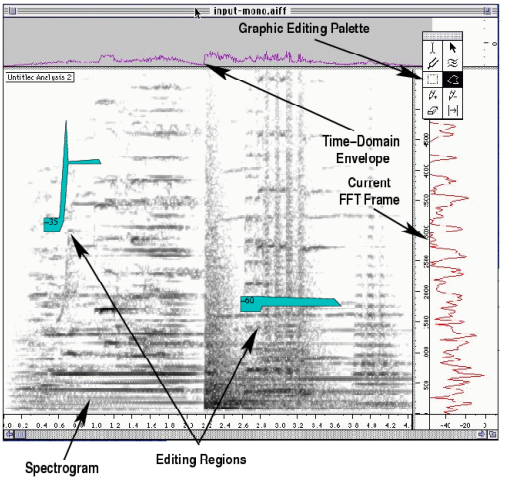

Figure 9: A typical *AudioSculpt* session

In fig. 9, some portions of the spectrogram have been delimited and different magnitude reductions have been applied to them.

Other applications, such as *Lemur* [22], (running on Apple Macintoshes) [33] or *Ceres* (developed by Oyvind Hammer at NoTam [23]) perform different sets of operations such as partial tracking and tracing, logical and algorithmic editing, timbre morphing, etc.

The contemporary sound designer can also benefit from tools which are specifically designed to transform sound objects in a controlled fashion. One

[21] http://www.ircam.fr
[22] http://www.cerlsoundgroup.org/Lemur/
[23] http://www.NoTam.uio.no/

such tool is *SMS* [24] (Spectral Modeling Synthesis), designed by Xavier Serra as an offspring of his and Smith's idea of analyzing sounds by decomposing them into stochastic and deterministic components [95] or, in other words, noise and sinusoids. SMS uses the Short-Time Fourier Transform (STFT) for analysis, tracking the most relevant peaks and resynthesizing from them the deterministic component of sound, while the stochastic component is obtained by subtraction. The decomposition allows flexible transformations of the analysis parameters, thus allowing good-quality time warping, pitch contouring, and sound morphing. In order to further improve the quality of transformations, extensions of the SMS model have been proposed though not included in the distributed software yet. Namely, a special treatment of transients has been devised as the way of getting rid of artifacts which can easily come into play when severe transformations are operated [108]. SMS comes with a very appealing graphical interface under Microsoft Windows, with a web-based interface, and is available as a command-line program for other operating systems, such as the various flavors of unix. SMS uses an implementation of the Spectral Description Interchange Format [25], which could potentially be used by other packages operating transformations based on the STFT. As an example, consider the following SMS synthesis score which takes the results of analysis and resynthesizes with application of a pitch-shifting envelope and an accentuation of inharmonicity:

```
InputSmsFile march.sms
OutputSoundFile exroc.snd
FreqSine 0 1.2 .5 1.1 .8 1 1 1
FreqSineStretch 0.2
```

B.5 Structure of a Digital Signal Processor

In this section we examine the ARES/MARS workstation as a prototypical case of hardware/software systems dedicated to digital audio processing. Namely, we explain the internal arithmetics of the X20 processor, the computational core of the workstation, and the memory management system.

We have mentioned that the ARES/MARS workstation uses an expansion board divided into two parts: a control part based on the microcontroller Motorola MC68302, and an audio processing part based on two proprietary X20

[24] http://www.iua.upf.es/~sms/
[25] http://cnmat.cnmat.Berkeley.edu/SDIF/

processors. The X20 processor runs, for each audio cycle, a 512-instruction microprogram, contained in a static external memory. Each microinstruction is 64 bits long, and it is computed in a 25ns machine cycle. Multiplying this cycle by the 512 instructions we get the working sampling rate of the machine, that is $F_s = 39062.5$Hz.

A rough scheme of the X20 processor is shown in figure 10, where we can notice three units:

- **Functional Unit**: adder (ALU), multiplier (MUL), registers (RM), data busses (C and Z);
- **Data Memory Unit**: data memories DMA and DMB, data busses (A, B, and W);
- **Control Unit**: addresses of data memories (ADR), access to external memory (FUN), connection to DAC/ADC audio bus, connection to microprogram memory and microcontroller (not shown in figure 10).

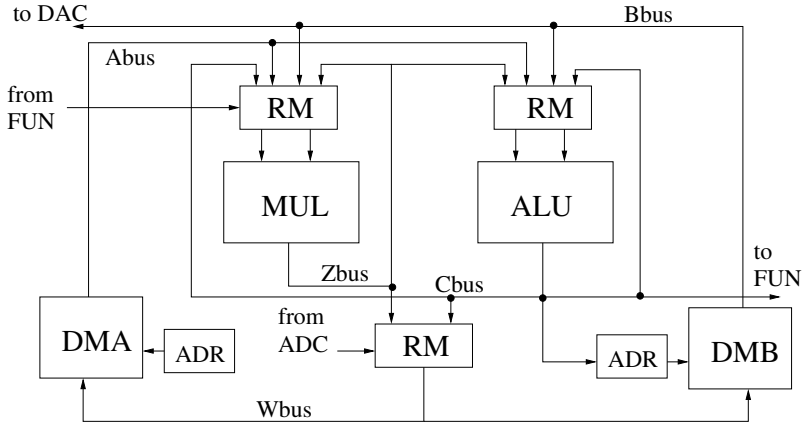

Figure 10: Block structure of the X20 processor

The computations are based on a circular data flow that involves the data memories and the functional unit. The presence of two data memories and one functional unit allows a parallel organization of microprograms. The data flow can be divided into four phases:

- Data gathering from memories DMA, DMB, or external memory (FUN);

- Selection of input data for the functional unit;
- Data processing by the functional unit;
- Insertion of the result back into the functional unit (by means of C and Z busses) or memorization into the data memories (W bus).

B.5.1 Memory Management

The waveforms, tables, samples, or delay lines, are allocated in the external memory[26], that is organized in, at most, 16 banks of 1MWord[27]. Each word is 16 bits long. In order to access the external memory we have to specify the base address in a 16-bit control word. Those bits are divided into two variable-length fields, separated by a zero bit. On the right there are ones, in a number n such that 32×2^n is the size of the table[28]. The field on the left is a binary number that denotes the ordinal number of the 32×2^n-words area allocated in memory. For instance, the control word $|0001|1101|1111|1111|$ ($1DFF_{16}$ in hexadecimal) represents the eight area of 16 KWords. Summarizing, in order to select an external table, the user has to specify the memory bank (0 to 15), the table size in powers of two, the offset, i.e., the ordinal number of table of the dimension that we are considering. The 16-bit control word is indeed only part of the 24-bit CWO register, the remaining 8 bits being used to select a waveform derived from reading the fourth part of a sine wave, memorized in 1024 words of internal read-only memory.

In another 24-bit register, called VAD, the table-reading phase pointer is stored. In order to access successive elements of the table, such register gets summed with the content of a 24-bit increment register. For example, 4KWord tables are accessed using an increment equal to 001000_{16}, while for 2KWord tables the increment is 002000_{16}. A 4KWord table is not stored in contiguous locations of the memory bank, but it uses locations that are seprated by $1024/4 = 0010_{16}$ positions. The first 2 bytes of the increment account for this distance. The extension of the phase to 3 bytes allows a fractional addressing, with interpolations between logically-contiguous samples. For instance, consider reading a 4KWord table: only the 12 most significant bits of the phase are used to address the table, the remaining 12 bits[29] being considered as the fractional part of the address and assigned to a register ALFA. If the 12 bits

[26] Called FUN or function memory
[27] 1MWord is equal to $2^{20} \approx 1000000$ words
[28] The minimal number of words in a table is, therefore, 32
[29] Actually, only the first 8.

of the phase give the value n, an interpolated read of the table will return the value [30]

$$y = (1 - ALFA)table(n) + ALFA\, table(n + 1) \quad (1)$$

With an increasing table size, the number of bits available for the fractional part decreases, and indeed this corresponds to a decrease in accuracy of interpolation for tables larger than 64KWord.

B.5.2 Internal Arithmetics

The data memories of the X20 processor are made of 24-bit locations, and 24 bits are also used for the registers feeding the ALU and for the busses C, Z, and W. On the other hand, we have only 16 bits for external functions and for the registers feeding the MUL. The internal arithmetics of the X20 can be summarized as:

- Representation of signals in two's complement fixed point, with normalization to one;
- Algebraic sum with 24-bit precision;
- Multiplication of two 16-bit numbers with 24-bit result;
- Tables and delay lines stored with 16-bit precision (FUN memory)
- 16-bit digital-to-analog and analog-to-digital conversion.

The addition can be performed as follows

- Normal mode: For the result, all the field of two's complement 24-bit numbers is used, with no handling of overflows.
  Ex.: $500000_{16} + 400000_{16} = 900000_{16} = -700000_{16}$.
- Overflow-protected mode: when an overflow occurs the result is set to the maximum or minimum representable number.
  Ex.: $500000_{16} + 400000_{16} = 900000_{16} = 7FFFFF_{16}$.
- Zero-protected mode: every negative result is forced to be zero.
- Overflow- and Zero-protected mode: the sum is first executed in overflow-protected mode, and any negative result is forced to be zero.

[30] The reader may observe that for ALFA equal to zero, the value $table(n)$ is returned, while for ALFA equal to one the returned value is $table(n + 1)$.

The first mode is useful whenever one has to generate cyclic waveforms or access to the memory cyclically, for instance to compute the phase pointer of an oscillator. The second mode is used when we are doing signal processing, since it protects from large-amplitude discontinuities and limit cycles (see section 1.6). The following table shows some examples of sums performed with the different modes[31]

a	b	a+b	a+b (OVP)	a+b (ZEP)	a+b (OVPZEP)
-0.5	0.7	0.2	0.2	0.2	0.2
0.5	0.7	-0.8	1.0	0.0	1.0
0.5	-0.7	-0.2	-0.2	0.0	0.0
-0.5	-0.7	0.8	-1.0	0.8	0.0

Multiplications are performed on the 16 most-significant bits of the operands in order to give a 24-bit result. The multiplication can be summarized in the following steps:

1) Consider only the 16 most-significant bits of the operands;

2) Multiply with 16-bit operand precision;

3) Consider only the 24 most-significant bits of the (31-bit) result.

The steps 1 and 3 imply quantization operations and precision loss. I passi 1 e 3 comportano delle operazioni di quantizzazione e pertanto comportano una perdita di precisione. The following table shows some examples of multiplications expressed in decimal and hexadecimal notations[32]

a	b	ab	a_{16}	b_{16}	ab_{16}
1.0	1.0	0.999939	7FFFFF	7FFFFF	7FFE00
1.0	0.5	0.499985	7FFFFF	400000	3FFF80
0.001	0.001	0.000001	0020C5	0020C5	000008
-1.0	1.0	-0.99970	800000	7FFFFF	800100
-1.0	-1.0	-1.0	800000	800000	800000

The examples highlight the need of looking at the results of multiplications with special care. The worst mistake is the one in the last line, where the result is off by 200% !

[31] Copied from the online help system of the ARES/MARS workstation.
[32] Copied from the online help system of the ARES/MARS workstation.

Another observation concerns the jump operations, that seem to be forbidden in an architecture that is based on the cyclic reading of a fixed number of microinstructions. Indeed, there are conditional instructions, that can change the selection of operands feeding the ALU according to a control value taken, for instance, from bus C. The presence of these instructions justify the name ALU for the adder, since it is indeed a Arithmetic Logic Unit.

B.5.3 The Pipeline

We have seen that the architecture of a Digital Signal Processor allows to perform some operations in parallel. For instance, we can simultaneously perform data transfers, multiplication, and addition. Most digital filters are based on the iterative repetition of operations such as

$$y = y + h_i s_i \tag{2}$$

where h_i are the coefficients of the filter and s_i are memory words containing the filter state. A DSP architecture such as the one of the X20 allows to specify, in a single microinstruction, the product of two registers containing h_i and s_i, the accumulation of the product obtained at the prior cycle into another register (containing y), and the register load with values h_i and s_i to be used at the next cycle. In other terms, the Multiply and Accumulate (MAC) operation is distributed onto three clock cycles, but for each cycle three MAC operations are in execution simultaneously. This is a realization of the principle of the pipeline, where the sample being "manufactured" has a latency time of three samples, but the frequency of sample delivery is one per clock cycle. In digital filters, another fundamental operation is the state update. In practice, after s_i has been used, it has to assume the value s_{i-1}. As it is shown in chapter 2, such operation can be avoided by proper indexing of memory accesses (circular buffering): Instead of moving the data with $s_i \leftarrow s_{i-1}$ we shift the indexes with $i \leftarrow i - 1$, in a circular fashion.

Appendix C

Fundamentals of psychoacoustics

Psychoacoustics is a "discipline within psychology concerned with sound, its perception and the physiological foundations of hearing" [75]. A few concepts and facts of psychoacoustics are certainly useful to the sound designer and to any computer scientist interested in working with sound. Several books provide a wider treatment of this topic, at different degrees of depth [86, 105, 42, 111].

C.1 The ear

The human ear is usually described as composed of three parts. This system is schematically depicted in figure 1.

the outer ear: The pinna couples the external space to the ear canal. Its shape is exploited by the hearing system to extract directional information from incoming sounds. The ear canal is a tube (length $l \approx 2.6$cm, diameter $d \approx 0.6$cm) closed on the inner side by a membrane called the ear drum. The tube acts as a quarter-of-wavelength resonator, exciting frequencies in the neighborhood of $f_0 = \frac{c}{4l} \approx 3.3$kHz, where c is the speed of sound in air;

the middle ear: It transmits mechanical energy, received from the ear drum, to the inner ear through a membrane called the oval window. To do so, it

uses a chain of small bones, called the hammer, the anvil, and the stirrup;

the inner ear: It is a cavity, called cochlea, shaped like a snail shell, which is shown rectified for clarity in figure 1. It contains a fluid and it is divided by the basilar membrane into two chambers: the scala vestibuli and the scala timpani. The length of the cochlea is about 3.5cm. Its diameter is about 2mm at the oval window (base) and it gets narrower at the other extreme (apex), where a narrow aperture (the helicotrema) allows the two chambers to communicate. On top of the basilar membrane, the tectorial membrane sustains about $16,000$ hair cells that pick up the transversal motion of the basilar membrane and transmit it to the brain.

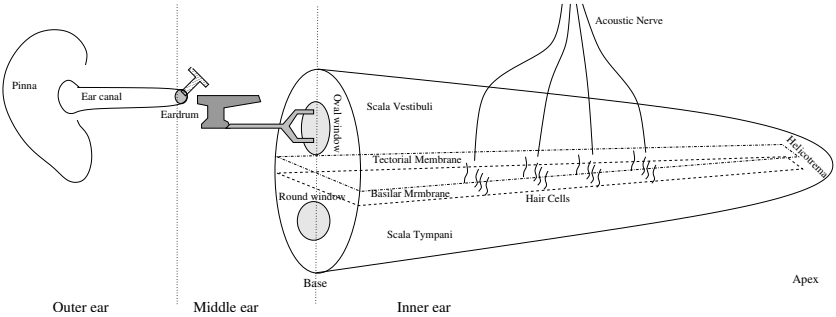

Figure 1: Cartoon physiology of the ear

The vibrations of the oval window excite the fluid of the scala vestibuli. By pressure differences between the scala vestibuli and scala timpani, the basilar membrane oscillates and transversal waves are propagated. The basilar membrane can be thought of as a string having a decreasing tension as we move from the base to the apex. This tension changes by about four orders of magnitude from base to apex. Along a string, the waves propagate at speed

$$c = \sqrt{\frac{T}{\rho_L}} = \sqrt{\frac{\text{Tension}}{\text{Linear density}}}\,, \tag{1}$$

and the wavelength associated with the component at frequency f is

$$\lambda = \frac{1}{f}\sqrt{\frac{T}{\rho_L}} = \frac{c}{f}\,. \tag{2}$$

The impedance of the string is $z_0 = \sqrt{\rho_L T}$ and, if v_{max} is the peak value of transversal velocity, the wave power is

$$P = \frac{1}{2} z_0 v_{max}^2 = \frac{1}{2} \sqrt{\rho_L T} v_{max}^2 \ . \tag{3}$$

While a wave component at frequency f is propagating from the base to the apex, its wavelength decreases (because tension decreases) and, due to the physical requirement of power constancy, its amplitude increases. However, this propagation is not lossless, and dissipation increases with the amplitude, so that a frequency-dependent maximum region will emerge along the basilar membrane (see figure 2). Since the high frequencies are more affected by propagation losses, their characteristic resonance areas are cluttered close to the base, while low frequencies are more widely distributed toward the apex. About two thirds of the length of the cochlea is devoted to low frequencies (about one fourth of the audio bandwidth), thus giving more frequency resolution to the slowly-varying components.

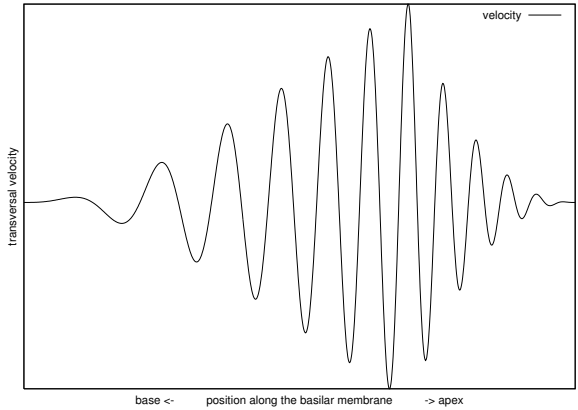

Figure 2: Cartoon of the transversal velocity pattern elicited by an incoming pure sine tone

C.2 Sound Intensity

Consider a sinusoidal point source in free space. It generates spherical pressure waves that carry energy. The acoustic intensity is the power by unit surface

that is carried by a wave front. It is a vectorial quantity having magnitude

$$I = \frac{p_{max}^2}{2}\frac{1}{z_0} = \frac{p_{max}^2}{2\rho c} = \frac{p_{RMS}^2}{\rho c} \ , \tag{4}$$

where p_{max} and p_{RMS} are the peak and root-mean-square (RMS) values of pressure wave, respectively, and $z_0 = \rho c =$ density $\times$ speed is the impedance of air.

At 1000Hz the human ear can detect sound intensities ranging from $I_{min} = 10^{-12}\mathrm{W/m^2}$ (threshold of hearing) to $I_{max} = 1\mathrm{W/m^2}$ (threshold of pain).

Consider two spherical shells of areas a_1 and a_2, at distances r_1 and r_2 from the point source. The lossless propagation of a wavefront implies that the intensities registered at the two distances are related to the areas by

$$I_1 a_1 = I_2 a_2 \ . \tag{5}$$

Since the area is proportional to the square of distance from the source, we also have

$$\frac{I_1}{I_2} = \left(\frac{r_2}{r_1}\right)^2 \ . \tag{6}$$

The intensity level is defined as

$$IL = 10 \log_{10} \frac{I}{I_0} \ , \tag{7}$$

where $I_0 = 10^{-12}\mathrm{W/m^2}$ is the sound intensity at the threshold of hearing. The intensity level is measured in decibel (dB), so that multiplications by a factor are turned into additions by an offset, as represented in table C.2. Similarly, the sound pressure level is defined as

$$SPL = 20 \log_{10} \frac{p_{max}}{p_{0,max}} = 20 \log_{10} \frac{p_{RMS}}{p_{0,RMS}} \tag{8}$$

where $p_{0,max}$ and $p_{0,RMS}$ are the peak and RMS pressure values at the threshold of hearing. For a propagating wave, we have that $IL = SPL$. For a standing wave, since there is no power transfer and since IL is a power-based measure, the SPL is more appropriate.

Given a reference tone with a certain value of IL at 1kHz, we can ask a subject to adjust the intensity of a probe tone at a different frequency until it matches the reference loudness perceptually. What we would obtain are the Fletcher-Munson curves, or equal-loudness curves, sketched in figure 3. Each

I	IL
$\times 1.26$	$+1$
$\times 2$	$+3$
$\times 10$	$+10$

Table C.1: Relation between factors in the linear intensity scale and shifts in the dB intensity-level scale

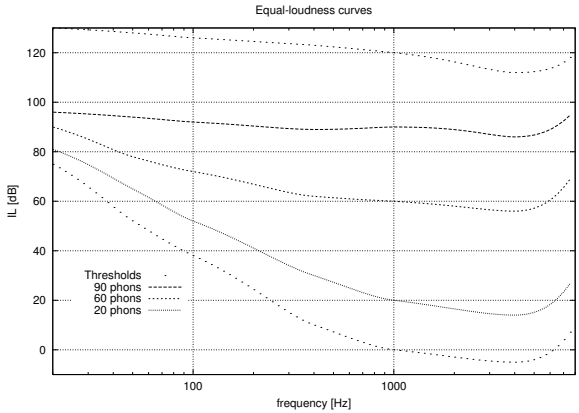

Figure 3: Equal-loudness curves. The parameters express values of loudness level in phons.

curve is parameterized on a value of loudness level (LL), measured in phons. The loudness level is coincident with the intensity level at 1kHz.

Even though the Fletcher-Munson curves are obtained by averaging the responses of human subjects, the LL is still a physical quantity, because it refers to the physical quantity IL and it does not represent the perceived loudness in absolute terms. In other words, doubling the loudness level does not mean doubling the perceived loudness. A genuine psychophysic measure is the loudness in sones, which can be obtained as a function of LL by asking listeners to compare sounds and decide when one sound is "twice as loud" as another. Somewhat arbitrarily, a LL of 40 phons is set equal to 1 sone. Figure 4 represents a possible average curve that may emerge from an experiment. The standardized loudness scale (ISO) uses the straight line approximation of figure 4,

that corresponds to the power law

$$L[\text{sones}] = \frac{1}{15.849}\left(\frac{I}{I_0}\right)^{0.3} . \tag{9}$$

Roughly speaking, an increment by 9 phons is needed to double the perceived subjective loudness in sones. This holds for tones at the same frequency or within the same critical band. In a physiological perspective, the critical band can be defined as the band of frequencies whose positions along the basilar membrane stay within the area excited by a single pure tone (see figure 2 and section C.4). We can say that the intensities of uncorrelated signals effectively sum:

$$I = I_1 + I_2 \; ; p^2 = p_1^2 + p_2^2 \Rightarrow p = \sqrt{p_1^2 + p_2^2} \, . \tag{10}$$

For uncorrelated pure tones within a critical band, if the law represented by the straight line in figure 4 does apply, if we double the intensity we have 3 phons of increment. Therefore, 3 doublings ($\times 8$) are needed to have an increase by 9 phons. This is the increase that roughly corresponds to a doubling in loudness. For example, 8 violins playing the same note at the same loudness level are needed to effectively double the perceived loudness.

If two sounds are far apart in frequency, their intensities sum much more effectively. In this case, using two sources at different frequencies also doubles the loudness.

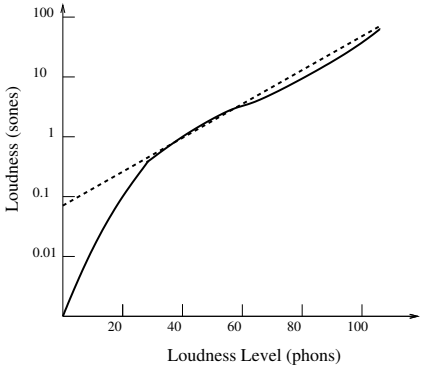

Figure 4: Sones vs. phons

Physics	Psychophysics
Physical Sound Φ	Perceived Sound Ψ
Intensity $I,\ \Delta I$	Loudness $L,\ \Delta L$
Frequency $f,\ \Delta f$	Pitch $p,\ \Delta p$
Duration $d,\ \Delta d$	Apparent Duration $\tilde{d},\ \Delta\tilde{d}$

Table C.2: Physics vs. Psychophysics

C.2.1 Psychophysics

In psychophysics, the Just Noticeable Difference (JND) of a physical quantity is the minimal difference of that quantity that can be noticed in two stimuli, or by modulation of a single stimulus. Since our perception is driven by neural firings statistically distributed in time, the appropriate way to measure JNDs is by subjective experimentation and statistical analysis. The random nature of perception is indeed the cause of JNDs, because the accuracy of our internal representations is limited by the intrinsic noise of these random processes.

The relation between physics and psychophysics is represented in table C.2.1 by means of three important acoustic quantities. The JNDs are represented by the symbol Δ preceding the physical or psychophysical variable name, in the latter case being a mnemonic for the internal noise variance.

The construction of psychophysical scales relies on the Fechner's idea[1] that:

> The value of the perceived quantity is obtained by counting the JNDs, and the result of such counting is the same whether we count physical or sensed JNDs. There is a "zero level" for sensation, i.e., the scale of sensations is a ratio scale (all four arithmetic operations are allowed).

For instance, for loudness:

$$\Delta L \cdot N_{JND} = L \ \Rightarrow N_{JND} = \frac{L}{\Delta L}\ . \tag{11}$$

If the JND is not constant:

$$N_{JND} = \int_0^L \frac{dL}{\Delta L(L)}\ . \tag{12}$$

[1] Gustav Theodor Fechner (1801-1887) is considered the father of psychophysics.

From the Fechner's idea we have

$$N_{JND} = \int \frac{dL}{\Delta L(L)} = \int \frac{dI}{\Delta I(I)} \ . \tag{13}$$

Fechner's psychophysics is based on two assumptions (exemplified for loudness):

1. ΔL is constant;

2. ΔI is proportional to I, or $\frac{\Delta I}{I} = k$, with k constant (Weber's law).

Based on the two assumptions, the Fechner's law is derived as

$$L = \Delta L \cdot N_{JND} = \Delta L \int \frac{dI}{kI} = \tilde{k} \log(I) \ , \tag{14}$$

for a certain value of the constant $\tilde{k}$.

For the loudness of pure tones neither the assumption 1 nor 2 are valid. Therefore, the Fechner's law (14) does not hold[2]. However, the Fechner's paradigm is the basis of new developments that provide models matching the experimental results quite closely. More details can be found in [42, 4].

Experimental curves similar to that reported in figure 4 show in many cases significant deviations from (14). For instance, the relation between intensity and loudness is more similar to

$$L \propto \sqrt[3]{I} \ , \tag{15}$$

as three doublings of intensity are needed for approximating one doubling in loudness.

Power laws such as the (15) are the natural outcome of the so called direct methods of psychophysical experimentation, where it is the sensation itself that is the unit for measuring other sensations. Such experimental paradigm was largely established by Stevens[3], and it is the one in use when the experimenter asks the subject to double or half the perceived loudness of a tone, or when a direct magnitude production or estimation is used.

[2] Weber's and Fechner's laws are taken for granted quite often in human-computer interaction.
[3] Stanley Smith Stevens (1906-1973).

C.3 Pitch

Periodic tones elicit a sensation of pitch, thus meaning that they can be ordered on a scale from low to high. Many aperiodic or even stochastic sounds can elicit pitch sensations, with different degrees of strength.

If we stick with pure tones for this section, pitch is the sensorial correlate of frequency, and it makes sense to measure the frequency JND using the tools of psychophysics. For instance, if a pure tone is slowly modulated in frequency we may seek for the threshold of modulation audibility. The resulting curve of average results would look similar to figure 5.

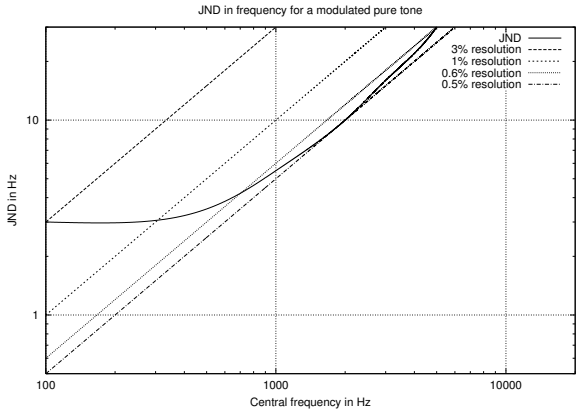

Figure 5: JND in frequency for a slowly modulated pure tone.

Again, from the curve of figure 5 we notice a significant deviation from the Weber's law $\Delta f \propto f$. The physiological interpretation is that there is more internal noise in the frequency detection in the very-low range.

If we integrate $\frac{1}{\Delta f(f)}$ we obtain a curve such as that of figure 6 that can be interpreted as a subjective scale for pitch, whose unit is called mel. Conventionally 1000 Hz corresponds to 1000 mel. This curve shouldn't be confused with the scales that organize musical height. Musical scales are based on the subdivision of the musical octave into a certain number of intervals. The musical octave is usually defined as the frequency range having the higher bound that has twice the value in Hertz of the first bound. On the other hand, the subjective scale for pitch measures the subjective pitch relationship between two sounds, and it is strictly connected with the spatial distribution of frequencies along the

basilar membrane. In musical reasoning, pitch is referred to as chroma, which is a different thing from the tonal height that is captured by figure 6.

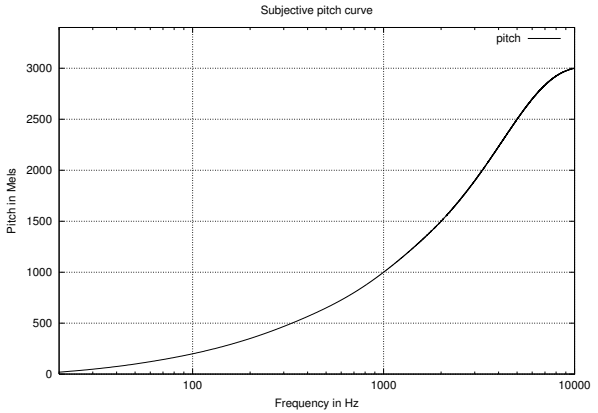

Figure 6: Subjective frequency curve, mel vs. Hz.

So far, we have described pitch phenomena referring to the position of hair cells that get excited along the basilar membrane. Indeed, the place theory of hearing is not sufficient to explain the accuracy of pitch perception and some intriguing effects such as the virtual pitch. In this effect, if a pure tone at frequency f_1 is superimposed to a pure tone at frequency $f_2 = \frac{3}{2} f_1$, the perceived pitch matches the missing fundamental at $f_0 = f_1/2$. If the reader, as an excercise, plots this superposition of waveforms, she may notice that the apparent periodicity of the resulting waveform is $1/f_0$. This indicates that a temporal processing of sound may occur at some stages of our perception. The hair cells convey signals to the fibers of the acoustic nerve. These neural contacts fire at a rate that depends on the transversal velocity of the basilar membrane and on its lateral displacement. The rate gets higher for displacements that go from the apex to the base, and this creates a periodicity in the firing rate that is multiple of the waveform periodicity. Therefore, the statistical distribution of neural spikes keeps track of the temporal behavior of the acoustic signals, and this may be useful at higher levels to extract periodicity information, for instance by autocorrelation processes [86].

Even for pure tones, pitch perception is a complex business. For instance, it is dependent on loudness and on the nature and quality of interfering sounds [42]. The pitch of complex tones is an overly complex topic to be discussed in this

appendix. It suffices to know that pitch perception of complex tones is linked to the third (after loudness and pitch) and most elusive attribute of sound, that is timbre.

C.4 Critical Band

As illustrated in figure 6 of chapter 2, two pure tones whose frequencies are close to each other give rise to the phenomenon of beating. In formula, from simple trigonometry

$$\sin \Omega_1 t + \sin \Omega_2 t = 2 \sin \frac{(\Omega_1 + \Omega_2)t}{2} \cos \frac{(\Omega_1 - \Omega_2)t}{2} , \qquad (16)$$

where the first sinusoidal term in the product can be interpreted as a carrier signal modulated by the second, cosinusoidal term.

As we vary the distance between the frequencies Ω_1 and Ω_2, the resulting sound is perceived differently, and a sense of roughness emerges for distances smaller than a certain threshold. A schematic view of the sensed signal is represented in figure 7. The solid lines may be interpreted as time-varying sensed pitch tracks. If they are far enough we perceive two tones. When they get closer, at a certain point a sensation of roughness emerges, but they are still resolved. As they get even closer, we stop perceiving two separate tones and, at a certain point, we hear a single tone that beats. Also, when they are very close to each other, the roughness sensation decreases.

The region where roughness gets in defines a critical band, and that frequency region roughly corresponds to the segment of basilar membrane that gets excited by the tone at frequency Ω_1. The sensation of roughness is related with that property of sound quality that is called consonance, and that can be evaluated along a continuous scale, as reported in figure 8. We notice that the maximum degree of dissonance is found at about one quarter of critical bandwidth.

C.5 Masking

When a sinusoidal tone impinges the outer ear, it propagates mechanically until the basilar membrane, where it affects the reception of other sinusoidal tones at nearby frequencies. If the incoming 400Hz tone, called the masker, has 70dB of IL, a tone at 600Hz has to be more than $30dB$ louder than its miniminal thresholding level in order to become audible in presence of the masker.

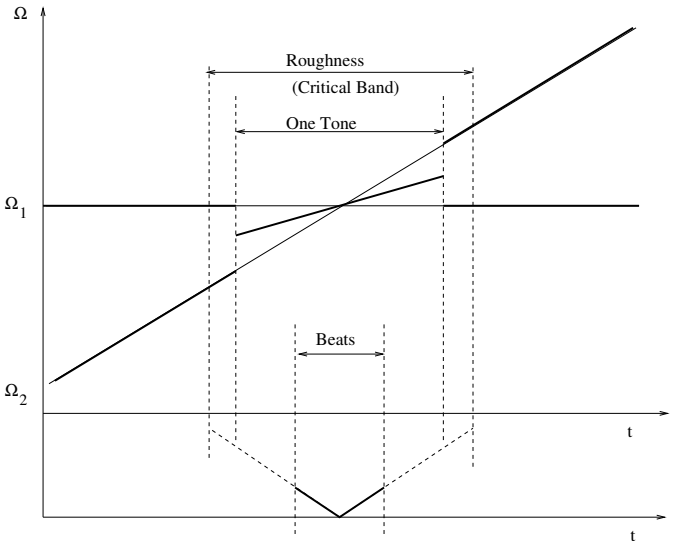

Figure 7: Schematic representation of the subjective phenomena of beats and roughness (adapted from [86])

This phenomenon is called masking and it is cartoonified in figure 9. Indeed, masking is ill-defined in the immediate proximity of the masker, because there the presence of beats may let the interference between masker and masked tone become apparent.

Two features of masking can be noticed in figure 9. First, masking is much more effective towards high frequencies (note also the log scale in frequency). Second, high-intensity maskers spread their effects even more towards high frequencies. The latter phenomenon is called upward spread of masking, and it is due to the nonlinear behavior of the outer hair cells of the cochlea, whose stiffness depends on the excitation they receive [4]. A high-frequency cell, excited by a lower-frequency tone, increases its stiffness and becomes less sensitive to components at its characteristic frequency.

In complex tones, the partials affect each other as far as masking is concerned, so that it may well happen that in a tone with a few dozens partials, only five or six emerge from a collective masking threshold. In a sound coding task, it is obvious that we should use all our resources (i.e., the bits) to encode those partials, thus neglecting the components that are masked. This idea is the

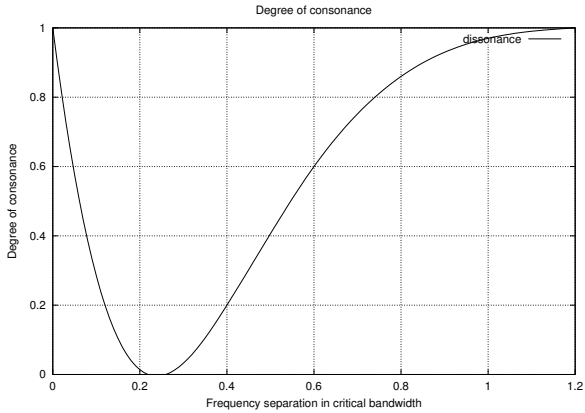

Figure 8: Degree of consonance between two sine tones as a function of their frequency distance, measured as a fraction of critical bandwidth (Measurement by Plomp and Levelt (1965) reported also in [105]).

basis for perceptual audio coding, as it is found in the MPEG-1 standard [69].

For coding purposes, it is also useful to look at temporal masking. Namely, the effects of masking extend in the future for up to 40ms (forward masking), and in the past for up to 10ms (backward masking). These temporal effects may occur because the brain integrates sound information over time, and there are inherent delays in this operation. Therefore, a soft tone preceding a louder tone by a couple of milliseconds is likely to be just canceled from our perceptual system.

C.6 Spatial sound perception

Classic psychoacoustic experiments showed that, when excited with simple sine waves, the hearing system uses two strong cues for estimating the apparent direction of a sound source. Namely, interaural intensity and time differences (IID and ITD) are jointly used to that purpose. IID is mainly useful above $1500Hz$, where the acoustic shadow produced by the head becomes effective, thus reducing the intensity of the waves reaching the contralateral ear. For this high-frequency range and for stationary waves, the ITD is also far less reliable, since it produces phase differences in sine waves which often exceed $360°$. Below $1500Hz$ the IID becomes smaller due to head diffraction which

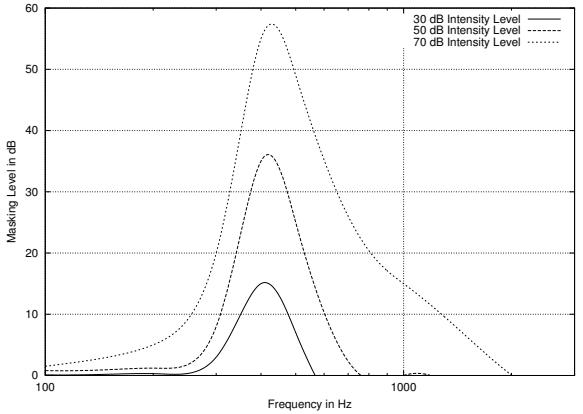

Figure 9: Schematic view of masking level for a sinusoidal masker at 400Hz at 30, 50, and 70 dB of intensity level.

overcomes the shadowing effect. In this low-frequency range it is possible to rely on phase differences produced by the ITD. IID and ITD can only partially explain the ability to discriminate among different spatial directions. In fact, if the sound source would move laterally along a circle (see figure 10) the IID and ITD would not change. The cone formed by the circle with the center of the head has been called cone of confusion. Front-back and vertical

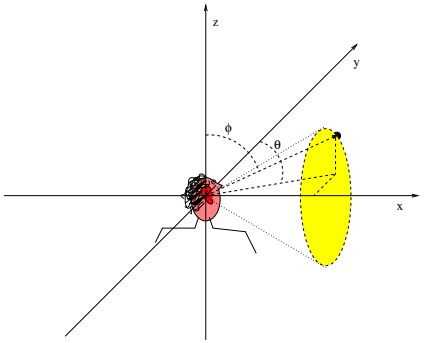

Figure 10: Interaural polar coordinate system and cone of confusion

discrimination within a cone of confusion are better understood in terms of

broadband signals and Head-Related Transfer Functions (HRTF). The system pinna - head - torso acts like a linear filter for a plane wave coming from a given direction. The magnitude and phase responses of this filter are very complex and direction dependent, so that it is possible for the listener to disambiguate between directions having the same, stationary, ITD and IID. In some cases, it is advantageous to think about these filtering effects in the time domain, thus considering the Head-Related Impulse Responses (HRIR) [13, 82].

Index

Bibliography

[1] M. Abramowitz and I. Stegun, editors. *Handbook of Mathematical Functions*. Dover Publications, New York, 1972.

[2] V. Algazi, R. Duda, D. Thompson, and C. Avendano. The CIPIC HRTF database. In *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 99–102, Mohonk, NY, Oct. 2001.

[3] J. Allen and D. Berkley. Image method for efficiently simulating small-room acoustics. *J. Acoustical Soc. of America*, 65(4):943–950, Apr. 1979.

[4] J. B. Allen. Psychoacoustics. In J. G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 422–437. John Wiley & Sons, 1999.

[5] X. Amatriain, J. Bonada, A. Loscos, and X. Serra. Spectral processing. In U. Zölzer, editor, *Digital Audio Effects*. John Wiley and Sons, Ltd., Chichester Sussex, UK, 2002.

[6] P. Andrenacci, F. Armani, R. Bessegato, A. Paladin, P. Pisani, A. Prestigiacomo, C. Rosati, S. Sapir, and M. Vetuschi. The new MARS workstation. In *Proc. International Computer Music Conference*, pages 215–219, Thessaloniki, Greece, Sept. 1997. ICMA.

[7] P. Andrenacci, E. Favreau, N. Larosa, A. Prestigiacomo, C. Rosati, and S. Sapir. MARS: RT20M/EDIT20 Development tools and graphical user interface for a sound generation board. In A. Strange, editor, *Proc. International Computer Music Conference*, pages 340–343, San Jose, CA, Oct. 1992. ICMA.

[8] D. Arfib. Digital synthesis of complex spectra by means of multiplication of non-linear distorted sine waves. *J. Audio Eng. Soc.*, 27(10):757–779, 1979.

[9] D. Arfib. Different ways to write digital audio effects programs. In *Proc. Conf. Digital Audio Effects (DAFx-98),* Barcelona, Spain, pages 188–191, Nov. 1998.

[10] D. Arfib, F. Keiler, and U. Zölzer. Source-filter processing. In U. Zölzer, editor, *Digital Audio Effects*, pages 299–372. John Wiley and Sons, Ltd., Chichester Sussex, UK, 2002.

[11] F. Armani, L. Bizzarri, E. Favreau, and A. Paladin. MARS - DSP Environment and Applications. In A. Strange, editor, *Proc. International Computer Music Conference*, pages 344–347, San Jose, CA, Oct. 1992. ICMA.

[12] A. Bernardi, G. Bugna, and G. D. Poli. Music signal analysis with chaos. In C. Roads, S. Pope, A. Picialli, and G. D. Poli, editors, *Musical Signal Processing*, pages 187–220. Swets & Zeitlinger, 1997.

[13] J. Blauert. *Spatial Hearing: the Psychophysics of Human Sound Localization.* MIT Press, Cambridge, MA, 1983.

[14] B. Blesser. An interdisciplinary synthesis of reverberation viewpoints. *J. Audio Eng. Soc.*, 49(10):867–903, 2001.

[15] G. Borin, G. De Poli, and A. Sarti. Sound Synthesis by Dynamic Systems Interaction. In D. Baggi, editor, *Readings in Computer-Generated Music*, pages 139–160. IEEE Computer Society Press, 1992.

[16] G. Borin, G. D. Poli, and D. Rocchesso. Elimination of delay-free loops in discrete-time models of nonlinear acoustic systems. *IEEE Transactions on Speech and Audio Processing*, 8(5):597–605, 2000.

[17] G. Borin, D. Rocchesso, and F. Scalcon. A physical piano model for music performance. In *Proc. International Computer Music Conference*, pages 350–353, Thessaloniki, Greece, Sept. 1997. ICMA.

[18] J. Borish. An Auditorium Simulation for Home Use. In *Audio Eng. Soc. Convention*, New York, 1983. AES.

[19] C. P. Brown and R. O. Duda. A structural model for binaural sound synthesis. *IEEE Trans. Speech and Audio Processing*, 6(5):476–488, Sept. 1998.

[20] C. Cadoz, A. Luciani, and J.-L. Florens. CORDIS-ANIMA: A modeling and simulation system for sound synthesis - the general formalism. *Computer Music J.*, 17(1):19–29, Spring 1993.

[21] S. Cavaliere, G. D. Giugno, and E. Guarino. MARS - The X20 device and the SM1000 board. In A. Strange, editor, *Proc. International Computer Music Conference*, pages 348–351, San Jose, CA, Oct. 1992. ICMA.

[22] A. Chaigne. On the Use of Finite Differences for Musical Synthesis. Application to Plucked Stringed Instruments. *J. Acoustique*, 5:181–211, 1992.

[23] J. M. Chowning. The synthesis of complex audio spectra by means of frequency modulation. *Journal of the Audio Eng. Soc.*, 21(7):526–534, 1973.

[24] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 1990.

[25] R. Courant and H. Robbins. *What is Mathematics?: an elementary approach to ideas and methods*. Oxford Un. Press, New York, 1941. Trad. It. Che Cos'è la Matematica?, Universale Scientifica Boringhieri, 1971.

[26] D. Gabor. Acoustical Quanta and the Theory of Hearing. *Nature*, 159(4044):591–594, May 1947.

[27] R. B. Dannenberg. Abstract time warping of compound events and signals. *Computer Music J.*, 21(3):61–70, 1997.

[28] R. B. Dannenberg. Machine tongues XIX: Nyquist, a language for composition and sound synthesis. *Computer Music J.*, 21(3):50–60, 1997.

[29] J. Dattorro. Effect design - part 1: Reverberator and other filters. *J. Audio Eng. Soc.*, 45(19):660–684, Sept. 1997.

[30] J. Dattorro. Effect design - part 2: Delay-line modulation and chorus. *J. Audio Eng. Soc.*, 45(10):764–788, Oct. 1997.

[31] G. De Poli and D. Rocchesso. Physically-based sound modeling. *Organised Sound*, 3(1):61–76, 1998.

[32] R. O. Duda and W. L. Martens. Range dependence of the response of a spherical head model. *J. Acoustical Soc. of America*, 104(5):3048–3058, Nov. 1998.

[33] K. Fitz and L. Haken. Sinusoidal modeling and manipulation using lemur. *Computer Music J.*, 20(4):44–59, 1997.

[34] F. Fontana and D. Rocchesso. Physical modeling of membranes for percussion instruments. *Acustica*, 84(13):529–542, Jan. 1998. S. Hirzel Verlag.

[35] A. Freed, X. Rodet, and P. Depalle. Synthesis and control of hundreds of sinusoidal partials on a desktop computer without custom hardware. In *Proc. 1993 Int. Computer Music Conf., Tokyo*, pages 98–101, 1993.

[36] B. Gardner and K. Martin. HRTF measurements of a KEMAR dummy-head microphone. Technical report # 280, MIT Media Lab, Cambridge, MA, 1994.

[37] W. G. Gardner. Efficient convolution without input-output delay. *J. Audio Eng. Soc.*, 43(3):127–136, 1995.

[38] W. G. Gardner. *3-D Audio using Loudspeakers*. Kluwer Academic Publishers, Norwell, MA, 1998.

[39] W. G. Gardner. Reverberation algorithms. In M. Kahrs and K. Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*, pages 85–131. Kluwer Academic Publishers, Norwell, MA, 1998.

[40] M. A. Gerzon. Unitary (Energy Preserving) Multichannel Networks with Feedback. *Electronics Letters V*, 12(11):278–279, 1976.

[41] W. M. Hartmann. Digital waveform generation by fractional addressing. *J. Acoustical Soc. of America*, 82(6):1883–1891, 1987.

[42] W. M. Hartmann. *Signals, Sound, and sensation*. Springer-Verlag, New York, 1998.

[43] D. A. Jaffe and J. O. Smith. Extensions of the Karplus-Strong Plucked String Algorithm. *Computer Music J.*, 7(2):56–69, 1983.

[44] J.-M. Jot. *Etude et Reálisation d'un Spatialisateur de Sons par Modèles Physiques et Perceptifs*. PhD thesis, TELECOM, Paris 92 E 019, 1992.

[45] J.-M. Jot and A. Chaigne. Digital Delay Networks for Designing Artificial Reverberators. In *Audio Eng. Soc. Convention*, Paris, France, Feb. 1991. AES.

[46] T. Kailath. *Linear Systems*. Prentice-Hall, Englewood Cliffs, 1980.

[47] K. Karplus and A. Strong. Digital Synthesis of Plucked String and Drum Timbres. *Computer Music J.*, 7(2):43–55, 1983.

[48] G. S. Kendall. A 3-D Sound Primer: Directional Hearing and Stereo Reproduction. *Computer Music J.*, 19(4):23–46, Winter 1995.

[49] G. Kuhn. Model for the interaural time differences in the azimuthal plane. *J. Acoustical Soc. of America*, 62:157–167, July 1977.

[50] H. Kuttruff. A Simple Iteration Scheme for the Computation of Decay Constants in Enclosures with Diffusely Reflecting Boundaries. *J. Acoustical Soc. of America*, 98(1):288–293, July 1995.

[51] T. I. Laakso, V. Välimäki, M. Karjalainen, and U. K. Laine. Splitting the Unit Delay—Tools for Fractional Delay Filter Design. *IEEE Signal Processing Magazine*, 13(1):30–60, Jan 1996.

[52] J. Laroche. Time and pitch scale modification of audio signals. In M. Kahrs and K. Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*, pages 279–309. Kluwer Academic Publishers, 1998.

[53] J. Makhoul. Linear prediction: A tutorial review. *Proc. IEEE*, 63(4):561–580, Apr. 1975.

[54] W. L. Martens. Psychophysical calibration for controlling the range of a virtual sound source: multidimensional complexity in spatial auditory display. In *Proc.Int. Conf. Auditory Display (ICAD-01)*, pages 197–207, Espoo, Finlnd, 2001.

[55] D. C. Massie. Wavetable sampling synthesis. In M. Kahrs and K. Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*, pages 311–341. Kluwer Academic Publishers, 1998.

[56] M. Mathews, J. E. Miller, F. R. Moore, J. R. Pierce, and J.-C. Risset. *The Technology of Computer Music*. MIT Press, Cambridge, MA, 1969.

[57] D. Mazzoni and R. Dannenberg. A fast data structure for disk-based audio editing. In *Proc. International Computer Music Conference*, La Habana, Cuba, Sep 2001. ICMA.

[58] S. K. Mitra. *Digital Signal Processing: A computer-Based Approach*. McGraw-Hill, New York, 1998.

[59] F. R. Moore. An Introduction to the Mathematics of Digital Signal Processing. Part I: Algebra, Trigonometry, and the Most Beautiful Formula in Mathematics. *Computer Music J.*, 2(1):38–47, 1978.

[60] F. R. Moore. A General Model for Spatial Processing of Sounds. *Computer Music J.*, 7(3):6–15, 1982.

[61] J. A. Moorer. About this Reverberation Business. *Computer Music J.*, 3(2):13–18, 1979.

[62] J. A. Moorer. The Manifold Joys of Conformal Mapping: Applications to Digital Filtering in the Studio. *J. Audio Eng. Soc.*, 31(11):826–840, 1983.

[63] P. M. Morse. *Vibration and Sound*. American Institute of Physics for the Acoustical Society of America, New York, 1991. 1st ed. 1936, 2nd ed. 1948.

[64] C. Müller-Tomfelde. Low-latency convolution for real-time applications. In *Proc. Audio Eng. Soc. Int. Conference*, pages 454–459, Rovaniemi, Finland, April 1999. Journal of the Audio Eng. Soc.

[65] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1989.

[66] A. V. Oppenheim and A. S. Willsky (with S. H. Nawab). *Signals and Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1997. Second edition.

[67] S. J. Orfanidis. *Introduction to Signal Processing*. Prentice Hall, Englewood Cliffs, N.J., 1996.

[68] J.-M. Pernaux, P. Boussard, and J.-M. Jot. Virtual sound source positioning and mixing in 5.1 implementation on the real-time system genesis. In *Proc. Conf. Digital Audio Effects (DAFx-98),* Barcelona, Spain, pages 76–80, Nov. 1998.

[69] K. C. Pohlmann. *Principles of Digital Audio*. McGraw-Hill, New York, 1995.

[70] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. Numerical recipes in c. 1988. sections available online at .

[71] M. Puckette. Pure data. In *Proc. International Computer Music Conference*, pages 224–227, Thessaloniki, Greece, Sept. 1997. ICMA.

[72] V. Pulkki. Virtual sound source positioning using vector base amplitude panning. *J. Audio Eng. Soc.*, 45(6):456–466, 1997.

[73] V. Pulkki, M. Karjalainen, and J. Huopaniemi. Analyzing virtual sound source attributes using binaural auditory models. *J. Audio Eng. Soc.*, 47(4):203–217, Apr. 1999.

[74] T. Quatieri and R. McAulay. Audio signal processing based on sinusoidal analysis/synthesis. In M. Kahrs and K. Brandenburg, editors, *Applications of Digital Signal Processing to Audio and Acoustics*, pages 343–416. Kluwer Academic Publishers, 1998.

[75] A. S. Reber and E. Reber. *The Penguin Dictionary of Psychology*. Penguin Books Ltd., London, UK, 2001. Third Edition.

[76] P. A. Regalia, S. K. Mitra, and P. P. Vaidyanathan. The Digital All-Pass Filter: A Versatile Signal Processing Building Block. *Proc. IEEE*, 76(1):19–37, Jan. 1988.

[77] C. Roads. Asynchronous granular synthesis. In *Representations of Musical Signals*, pages 143–186. MIT Press, Cambridge, MA, 1991.

[78] C. Roads. *The Computer Music Tutorial*. MIT Press, Cambridge, Mass., 1996.

[79] D. Rocchesso. The Ball within the Box: a sound-processing metaphor. *Computer Music J.*, 19(4):47–57, Winter 1995.

[80] D. Rocchesso. *Strutture ed Algoritmi per l'Elaborazione del Suono basati su Reti di Linee di Ritardo Interconnesse*. Phd thesis, Università di Padova, Dipartimento di Elettronica e Informatica, Feb. 1996.

[81] D. Rocchesso. Maximally-Diffusive yet Efficient Feedback Delay Networks for Artificial Reverberation. *IEEE Signal Processing Letters*, 4(9):252–255, Sept. 1997.

[82] D. Rocchesso. Spatial effects. In U. Zölzer, editor, *Digital Audio Effects*, pages 137–200. John Wiley and Sons, Ltd., Chichester Sussex, UK, 2002.

[83] D. Rocchesso and F. Scalcon. Bandwidth of perceived inharmonicity for physical modeling of dispersive strings. *IEEE Transactions on Speech and Audio Processing*, 7(5):597–601, Sept. 1999.

[84] D. Rocchesso and J. O. Smith. Circulant and Elliptic Feedback Delay Networks for Artificial Reverberation. *IEEE Transactions on Speech and Audio Processing*, 5(1):51–63, Jan. 1997.

[85] D. Rocchesso and J. O. Smith. Generalized digital waveguide networks. *IEEE Transactions on Speech and Audio Processing*, 11(5), 2003.

[86] J. G. Roederer. *Introduction to the Physics and Psychophysics of Music*. Springer-Verlag, Heidelberg, 1975.

[87] B. Schottstaedt. Machine tongues XVII: CLM: Music V meets common lisp. *Computer Music J.*, 18(2):30–37, 1994.

[88] M. R. Schroeder. Improved Quasi-Stereophony and "Colorless" Artificial Reverberation. *J. Acoustical Soc. of America*, 33(8):1061–1064, Aug. 1961.

[89] M. R. Schroeder. Natural-Sounding Artificial Reverberation. *J. Audio Eng. Soc.*, 10(3):219–233, July 1962.

[90] M. R. Schroeder. Digital Simulation of Sound Transmission in Reverberant Spaces. *J. Acoustical Soc. of America*, 47(2):424–431, 1970.

[91] M. R. Schroeder. Computer Models for Concert Hall Acoustics. *American Journal of Physics*, 41:461–471, 1973.

[92] M. R. Schroeder. *Computer Speech: Recognition, Compression, and Synthesis*. Springer Verlag, Berlin, Germany, 1999.

[93] M. R. Schroeder and B. Logan. "Colorless" Artificial Reverberation. *J. Audio Eng. Soc.*, 9:192–197, July 1961. reprinted in the IRE Trans. on Audio.

[94] X. Serra. Musical sound modeling with sinusoids plus noise. In C. Roads, S. Pope, A. Picialli, and G. D. Poli, editors, *Musical Signal Processing*, pages 91–122. Swets & Zeitlinger, 1997.

[95] X. Serra and J. O. Smith. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14(4):12–24, 1990.

[96] J. O. Smith. An allpass approach to digital phasing and flanging. In *Proc. International Computer Music Conference*, page 236, Paris, France, 1984. ICMA. Also available as Rep. STAN-M-21, CCRMA, Stanford University.

[97] J. O. Smith. Fundamentals of Digital Filter Theory. *Computer Music J.*, 9(3):13–23, 1985.

[98] J. O. Smith. Physical modeling using digital waveguides. *Computer Music J.*, 16(4):74–91, Winter 1992.

[99] J. O. Smith and J. S. Abel. The Bark Bilinear Transform. In *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, Mohonk, NY, Oct. 1995.

[100] J. O. Smith and B. Friedlander. Adaptive interpolated time-delay estimation. *IEEE Trans. Aerospace and Electronic Systems*, 21(2):180–199, Mar. 1985.

[101] J. Stautner and M. Puckette. Designing Multichannel Reverberators. *Computer Music J.*, 6(1):52–65, Spring 1982.

[102] K. Steiglitz. *A Digital Signal Processing Primer*. Addison-Wesley, Menlo Park, CA, 1996.

[103] J. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth & Brooks, Pacific Grove, CA, 1989.

[104] C. R. Sullivan. Extending the Karplus-Strong Algorithm to Synthesize Electric Guitar Timbres with Distortion and Feedback. *Computer Music J.*, 14(3):26–37, 1990.

[105] J. Sundberg. *The Science of Musical Sounds*. Academic Press, San Diego, CA, 1989. First Ed. 1973.

[106] B. Vercoe. Csound: A manual for the audio processing system and supporting programs with tutorials. Technical report, Media Lab, M.I.T., Cambridge, Massachusetts. Software and Manuals available from ftp://ftp.maths.bath.ac.uk/pub/dream/, 1993.

[107] B. L. Vercoe, W. G. Gardner, and E. D. Scheirer. Structured Audio: Creation, Transmission, and Rendering of Parametric Sound Representations. *Proc. IEEE*, 86(5):922–940, May 1998.

[108] T. S. Verma, S. N. Levine, and T. H. Y. Meng. Transient modeling synthesis: a flexible analysis/synthesis tool for transient signals. In *Proc. International Computer Music Conference*, pages 164–167, Thessaloniki, Greece, Sept. 1997. ICMA.

[109] U. Zölzer. *Digital Audio Signal Processing*. John Wiley and Sons, Inc., Chichester, England, 1997.

[110] U. Zölzer, editor. *Digital Audio Effects*. John Wiley and Sons, Ltd., Chichester Sussex, UK, 2002.

[111] E. Zwicker and H. Fastl. *Psychoacoustics: Facts and Models*. Springer Verlag, Berlin, Germany, 1990.